

DEFENCE



DÉFENSE

## **Reconfigurable Digital IIR and FIR Filters**

B. Gosselin and C. Wilcox

**DISTRIBUTION STATEMENT A**  
Approved for Public Release  
Distribution Unlimited

**Defence R&D Canada**

TECHNICAL REPORT

DREO TR 2001-099

November 2001



National  
Defence

Défense  
nationale

**Canada**

**20020305 148**

© Her Majesty the Queen as represented by the Minister of National Defence, 2001

© Sa majesté la reine, représentée par le ministre de la Défense nationale, 2001

## ABSTRACT

The work presented in this document contributes to the ROBR (Reconfigurable Omni Band Radio) project started by the Defence Research Establishment Ottawa and the Communication Research Centre in 1997. ROBR is a testbed implementation of a reconfigurable satellite communications (satcom) terminal that makes use of a software communications architecture. Such a system can enable the use of a single ground terminal to communicate over multiple satellite communications or terrestrial links by supporting multiple standards. The ROBR hardware architecture includes a microprocessor and several digital signal processor (DSP) boards. The objective of this report is to document the work done to provide a set of reconfigurable digital filters for use in the ROBR. Five infinite impulse response (IIR) filtering modules and four finite impulse response (FIR) filtering modules have been implemented. The function of these modules is to compute the coefficients of a desired filter design. Also, IIR and FIR signal processing modules have been implemented to process digital signals using the computed coefficients. The modules have been implemented in the C programming language and are targeted for use on a DSP chip. The implementation of the modules has been verified and compared with the results obtained with the *Signal Processing* toolbox from MATLAB.

## RÉSUMÉ

Les travaux présentés dans le présent document contribuent au projet ROBR (radio omni-bande reconfigurable) entrepris par le Centre de recherches pour la défense, Ottawa et le Centre de recherches sur les communications en 1997. Il s'agit de la mise en œuvre d'un prototype de terminal reconfigurable de télécommunications par satellite qui a recours à une architecture logicielle pour les communications. Ce terminal peut permettre l'utilisation d'un seul terminal au sol pour assurer des communications au moyen de plusieurs liaisons de communications par satellite ou de plusieurs liaisons de Terre en vertu de plusieurs normes. L'architecture matérielle du ROBR comprend un microprocesseur et plusieurs cartes de traitement numérique des signaux (DSP). Le présent rapport a pour but de documenter le travail effectué pour la fourniture d'un jeu de filtres numériques reconfigurables en vue de son utilisation dans le ROBR. Cinq modules de filtrage des réponses impulsionnelles infinies (RII) et quatre modules de filtrage des réponses impulsionnelles finies (RIF) ont été mis au point. Ces modules servent au calcul des coefficients de la conception désirée des filtres. Des modules de traitement des signaux RII et RIF ont aussi été mis au point pour le traitement numérique des signaux au moyen des coefficients calculés. Ces modules, configurés en langage de programmation C, doivent être utilisés sur des puces DSP. Le fonctionnement des modules a été vérifié et comparé aux résultats obtenus au moyen du produit *Signal Processing Toolbox* de MATLAB.

## EXECUTIVE SUMMARY

The work presented in this document contributes to the ROBR (Reconfigurable Omni Band Radio) project started by the Defence Research Establishment Ottawa and the Communication Research Centre in 1997. ROBR is a testbed implementation of a reconfigurable satellite communications (satcom) terminal that can support multiple standards. Such a system potentially allows the Canadian Forces to use a single ground terminal to communicate over multiple satellite communications or terrestrial links.

The ROBR hardware architecture includes a microprocessor and several digital signal processor (DSP) boards. The objective of this report is to document the work done to provide a set of reconfigurable digital filters for use in the ROBR. Five infinite impulse response (IIR) filtering modules and four finite impulse response (FIR) filtering modules have been implemented. These modules will be integrated in the DSPs and processed by them. The function of these modules is to compute the coefficients of a desired filter design. Also, IIR and FIR signal processing modules have been implemented to process test input signals using the filter coefficients generated.

The implemented techniques to design IIR filters are based on transformation of continuous-time IIR systems into discrete-time IIR systems. Two conversion methods have been implemented to produce a discrete filter design from an analog filter design: the "Bilinear Transformation" and the "Impulse Invariance" methods. Five major types of IIR filters have been implemented, the Butterworth filter, the Chebyshev filter, the inverse Chebyshev filter, the Elliptical filter and the Bessel filter. The computation of the coefficients for the IIR filtering modules is separated into three steps: the analog design computation, the conversion of the analog design into a discrete design and the factorization step which produces the coefficients of the digital filter.

The implemented techniques for computing the coefficients of the FIR filtering modules are the "Frequency Sampling Design" method, the "Design by Windowing" method and the "Parks-McClellan" method (also called the "Remez Exchange Algorithm"). The general procedure for FIR filter design is to sample the frequency response of a filter and then compute the inverse discrete Fourier transform (IDFT).

An FIR implementation to compute the coefficients of a Gaussian filter has been implemented. The method used to calculate the coefficients of this filter is the "Frequency Sampling Design" method. Finally, a digital integrator was implemented. The function of a digital integrator is to sum a digital input sequence over time. The Gaussian filter module and the digital integrator module may be used in the premodulation stage of a Gaussian minimum shift keying (GMSK) digital modulator for the ROBR.

The modules have been implemented in the C language for further implementation on a DSP chip. The implementation of both IIR and FIR filter modules have been verified and compared with the results obtained with the *Signal Processing* toolbox from MATLAB.



The implementation of two filtering modules has been successfully done on a TMS320c6201 digital signal processor from Texas Instruments mounted on a Daytona DSP board from Spectrum.

Gosselin, B., and Wilcox, C., 2001, Reconfigurable Digital IIR and FIR Filters, DREO TR 2001-099, Defence Research Establishment Ottawa.

## SOMMAIRE

Les travaux présentés dans le présent document contribuent au projet ROBR (radio omni-bande reconfigurable) entrepris par le Centre de recherches pour la défense, Ottawa et le Centre de recherches sur les communications en 1997. Il s'agit de la mise en œuvre d'un prototype de terminal reconfigurable de télécommunications par satellite à l'appui de plusieurs normes. Ce terminal pourrait permettre aux Forces canadiennes d'utiliser un seul terminal au sol pour assurer des communications au moyen de plusieurs liaisons de communications par satellite ou de plusieurs liaisons de Terre.

L'architecture matérielle du ROBR comprend un microprocesseur et plusieurs cartes de processeur numérique de signaux (DSP). Le présent rapport a pour but de documenter le travail effectué pour la fourniture d'un jeu de filtres numériques reconfigurables en vue de son utilisation dans le ROBR. Cinq modules de filtrage des réponses impulsionnelles infinies (RII) et quatre modules de filtrage des réponses impulsionnelles finies (RIF) ont été mis au point. Ces modules seront intégrés aux DSP et traités par eux. Ils servent au calcul des coefficients de la conception désirée des filtres. Des modules de traitement des signaux RII et RIF ont aussi été mis au point pour le traitement des signaux d'entrée d'essai au moyen des coefficients générés pour les filtres.

Les techniques utilisées pour la conception des filtres RII sont fondées sur la transformation des systèmes RII à temps continu en systèmes RII à temps discret. Deux méthodes de conversion ont été mises en œuvre pour la production d'une conception de filtre discret à partir de la conception d'un filtre analogique : la méthode de "transformation bilinéaire" et la méthode "par invariance impulsionnelle". Cinq grands types de filtres RII ont été sélectionnés : le filtre de Butterworth, le filtre de Chebyshev, le filtre de Chebyshev inverse, le filtre elliptique et le filtre de Bessel. Le calcul des coefficients pour les modules de filtrage RII se divise en trois étapes : le calcul de la conception analogique, la conversion de la conception analogique en conception discrète et la factorisation (qui produira les coefficients du filtre numérique).

Les techniques sélectionnées pour le calcul des coefficients des modules de filtrage RIF sont la méthode de "conception par échantillonnage de fréquences", la méthode de "conception par fenêtrage" et la méthode de "l'algorithme d'optimisation de Parks-McClellan" (aussi appelée la "fonction de Remez"). La procédure générale de conception des filtres RIF consiste à échantillonner la réponse en fréquence d'un filtre, puis à calculer la transformée de Fourier discrète inverse.

Une application aux RIF pour le calcul des coefficients d'un filtre gaussien a été effectuée. La méthode utilisée pour le calcul des coefficients de ce filtre est la méthode de "conception par échantillonnage de fréquences". Enfin, un module d'intégrateur numérique a été élaboré. Un intégrateur numérique sert au calcul de la somme d'une séquence d'entrée numérique pour une période donnée. Le module de filtre gaussien et le module d'intégrateur numérique peuvent être utilisés à l'étape de la prémodulation d'un modulateur numérique de modulation par déplacement minimal avec filtrage gaussien (MDMG).

Les modules, configurés en langage de programmation C, doivent être utilisés sur des puces DSP à une date ultérieure. Trois modules de filtrage ont été implantés avec succès sur un processeur numérique de signaux TMS320c6201 de Texas Instruments monté sur une carte DSP Daytona de Spectrum. Le fonctionnement des modules a été vérifié et comparé aux résultats obtenus au moyen du produit *Signal Processing Toolbox* de MATLAB.

Gosselin, B., et Wilcox, C., 2001, Reconfigurable Digital IIR and FIR Filters, DREO TR 2001-099, Centre de recherches pour la défense Ottawa.

# TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT .....	iii
RÉSUMÉ .....	iii
EXECUTIVE SUMMARY .....	v
SOMMAIRE .....	vii
TABLE OF CONTENTS .....	ix
LIST OF SYMBOLS.....	xi
LIST OF ABBREVIATIONS .....	xi
LIST OF FIGURES AND TABLES.....	xiii
<b>1 INTRODUCTION.....</b>	<b>1</b>
1.1 SCOPE .....	1
<b>2 DIGITAL FILTERS .....</b>	<b>3</b>
2.1 DISCRETE-TIME SIGNAL .....	3
2.2 LINEAR TIME-INVARIANT (LTI) SYSTEMS .....	4
2.3 THE LAPLACE TRANSFORM AND THE Z-TRANSFORM.....	4
2.4 FILTER DESIGN AND MODULE SPECIFICATIONS .....	6
<b>3 INFINITE IMPULSE RESPONSE FILTERS .....</b>	<b>9</b>
3.1 INFINITE IMPULSE RESPONSE (IIR) LTI SYSTEMS.....	9
3.1.1 <i>Flow diagrams of recursive structures</i> .....	9
3.2 IIR FILTER DESIGN .....	10
3.2.1 <i>Analog filter design</i> .....	10
3.2.1.1 Classical analog filter approximations .....	10
3.2.1.2 Butterworth filter properties.....	11
3.2.1.3 Chebyshev filter properties .....	12
3.2.1.4 Inverse Chebyshev properties .....	14
3.2.1.5 Elliptical filter properties .....	16
3.2.1.6 Bessel filter properties.....	19
3.2.2 <i>Conversion of analog IIR filters for digital implementation</i> .....	20
3.2.2.1 Bilinear transformation .....	20
3.2.2.2 Impulse invariance method .....	22
3.3 IIR FILTERING MODULE IMPLEMENTATION .....	24
3.3.1 <i>System</i> .....	24
3.3.2 <i>IIR coefficients computation modules</i> .....	25
3.3.2.1 Analog design computation.....	26
3.3.2.2 Analog-to-digital Conversion.....	27
3.3.2.3 Factorization .....	27
3.3.3 <i>IIR signal processing module</i> .....	29
<b>4 FINITE IMPULSE RESPONSE FILTERS.....</b>	<b>31</b>
4.1 FIR LTI SYTEMS.....	31
4.1.1 <i>Flow diagrams for non-recursive structures</i> .....	31

4.2	FIR FILTER DESIGN .....	32
4.2.1	<i>"Frequency Sampling Design" filter module</i> .....	32
4.2.1.1	Gibbs phenomenon .....	33
4.2.2	<i>"Design by Windowing" filter module</i> .....	34
4.2.3	<i>Parks-McClellan filter module</i> .....	37
4.2.3.1	Chebyshev approximation .....	37
4.2.3.2	Alternation theorem .....	39
4.2.4	<i>Gaussian digital filter module</i> .....	41
4.2.4.1	Digital integrator module .....	43
4.3	FIR FILTER MODULE IMPLEMENTATION .....	43
4.3.1	<i>System</i> .....	43
4.3.2	<i>FIR coefficients computation modules</i> .....	44
4.3.2.1	Frequency sampling design method module implementation .....	44
4.3.2.2	Parks-McClellan method implementation .....	47
4.3.2.3	Gaussian filter implementation .....	48
4.3.3	<i>FIR signal processing modules</i> .....	49
4.3.4	<i>Use of the filter and signal processing modules</i> .....	50
5	DSP IMPLEMENTATION .....	53
5.1	EXCHANGING DATA BETWEEN THE DAYTONA AND THE HOST STATION .....	53
5.2	STATIC AND DETERMINED LENGTH MEMORY ALLOCATION .....	54
5.3	DYNAMIC MEMORY ALLOCATION .....	55
5.4	HOST PROGRAM .....	57
5.4.1	<i>Host software functions</i> .....	57
5.5	DSP PROGRAM .....	57
5.5.1	<i>DSP software functions</i> .....	57
6	RESULTS AND VERIFICATION .....	59
6.1	METHODOLOGY .....	59
6.2	RESULTS .....	60
6.2.1	<i>IIR filter module verification</i> .....	60
6.2.2	<i>FIR filter module verification</i> .....	61
7	SUMMARY .....	65
8	REFERENCES .....	67
APPENDICES		
APPENDIX A .....		A1
APPENDIX B .....		B1
APPENDIX C .....		C1

## LIST OF SYMBOLS

$T$	Sampling period
$F_s$	Sampling frequency
$\omega$	Digital frequency
$\omega_c$	Digital cutoff frequency
$\Omega$	Analog frequency
$\Omega_c$	Analog cutoff frequency
$p_k$	Poles of a transfer function
$z_k$	Zeroes of a transfer function
$h_c(t)$	Continuous-time transfer function or impulse response
$H_c(\Omega)$	Continuous frequency transfer function
$h[n]$	Discrete-time transfer function
$H(\omega)$	Discrete-frequency transfer function
$\delta[n]$	Impulse signal
$H_0$	Static gain of a filter
$n$	Filter order

## LIST OF ABBREVIATIONS

ROBR	Reconfigurable OmniBand Radio
LTI	Linear Time-Invariant
DTFT	Discrete-Time Fourier Transform
DFT	Discrete Fourier Transform
IDFT	Inverse Discrete Fourier Transform
IIR	Infinite Impulse Response
FIR	Finite Impulse Response
FFT	Fast Fourier Transform
GMSK	Gaussian Minimum Shift Keying

## LIST OF FIGURES AND TABLES

	<u>Page</u>
Figure 1 Graphical representation of a discrete-time signal .....	3
Figure 2 Representation of a Linear Time-Invariant System.....	4
Figure 3 Magnitude response for a low-pass filter .....	6
Figure 4 Flow diagram implementing the Direct Form I realization of an IIR filter.....	9
Figure 5 Frequency response for a Butterworth filter.....	12
Figure 6 Frequency response for a Chebyshev filter .....	13
Figure 7 Frequency response for Inverse Chebyshev filters.....	15
Figure 8 Frequency response for an Elliptical filter .....	17
Figure 9 Frequency response for a Bessel filter for different values of filter order, n .....	19
Figure 10 Mapping of continuous frequency to digital frequency.....	22
Figure 11 Block diagram of the implementation of the reconfigurable IIR filtering modules .....	24
Figure 12 Cutoff frequency normalization and analog frequency computation.....	25
Figure 13 Chebyshev analog design computation algorithm.....	26
Figure 14 Implementation of the Bessel analog design computation block as a lookup table .....	27
Figure 15 Bilinear transform analog-to-digital conversion algorithm .....	28
Figure 16 Expansion formula algorithm .....	28
Figure 17 IIR signal processing module algorithm.....	30
Figure 18 Flow diagram of the Direct Form realization of an FIR filter .....	31
Figure 19 The desired frequency response $H_d(\omega)$ of an ideal low-pass filter .....	32
Figure 20 (a) Frequency response of an FIR filter affected by Gibbs phenomenon (b) Power Spectrum of an FIR filter affected by Gibbs phenomenon.....	34
Figure 21 Commonly used windows .....	36
Figure 22 Comparison between the “Frequency Sampling Design” and “Design by Windowing” methods for FIR filter design. (a) Frequency response. (b) Power spectrum.....	37
Figure 23 The desired frequency response $D(\omega)$ of a low-pass filter .....	38
Figure 24 Typical example of a low-pass filter approximation that is optimal according to the alternation theorem for $c = 7$ .....	41
Figure 25 The premodulator stage of GMSK digital modulator includes a digital integrator followed by a Gaussian filter .....	42
Figure 26 Frequency response for a Gaussian filter .....	42
Figure 27 Block diagram of FIR filter module implementation.....	43
Figure 28 Computation of the number of samples to include in the passband of an ideal low-pass filter.....	45
Figure 29 Sampling of an ideal low-pass filter frequency response .....	45
Figure 30 Inverse discrete Fourier transform algorithm performed on the ideal low-pass filter frequency response samples .....	46
Figure 31 “Design by Windowing” algorithm.....	47
Figure 32 Content of the file <i>remezex.c</i> .....	48
Figure 33 Sampling algorithm of a Gaussian distribution.....	49

Figure 34	FIR signal processing module algorithm.....	50
Figure 35	Command line for Butterworth filter module.....	51
Figure 36	Example of an output file generated by the coefficients computation modules...	51
Figure 37	Command line for executing an IIR signal processing module. ....	52
Figure 38	Example of an output file generated by a signal processing module .....	52
Figure 39	Static allocation of the variables in the SDRAM memory .....	54
Figure 40	Flow diagram of the implementation of the filter modules.....	55
Figure 41	Sysmem memory section allocation in the DSP memory map. ....	56
Figure 42	Impulse signal generated with Matlab.....	60
Figure 43	Error curve for reconfigurable filter using the “Frequency Sampling Design” method. (a) for N=10 coefficients. (b) for N=30 coefficients.....	61

Table 1	List of specifications for a low-pass filter .....	6
Table 2	Computational requirements for Direct Form I structure .....	10
Table 3	IIR Filter types implemented by the corresponding modules.....	25
Table 4	Computational requirements of the Direct Form I structure of FIR filters.....	32
Table 5	Inverse Discrete Fourier Transform formulas for FIR Design .....	33
Table 6	Commonly used window functions .....	36
Table 7	FIR filter types implement by the corresponding modules.....	44
Table 8	Memory configuration of the TMS320C6201 .....	53
Table 9	Corresponding MATLAB functions for the IIR modules verification.....	59
Table 10	Corresponding MATLAB functions for the IIR modules verification .....	59



# 1 INTRODUCTION

With the rapid increase of communications services and systems being developed and implemented using different standards, much effort has been directed over the past decade to deal with issues relating to interoperability and compatibility of these various systems. While global standardization can be one solution to the problem, a more practical remedy may be to develop transceivers that can support different frequency bands and different waveform standards using a common hardware platform or device. A feasible way to achieve this is to implement the waveform standards in software to provide the required flexibility.

Although initially focused on the personal communications services industry, the software radio concept can be extended to other applications such as satellite communications (satcom). Just as multiple standards are being considered for integration on a single device (e.g. cellular phone), multiple satcom waveforms can be implemented in software to run on a single hardware platform or terminal. As with personal communications services, this approach potentially offers benefits for interoperability, ease of future upgrades, and even integration of future systems.

Defence Research Establishment Ottawa (DREO) is engaged in a project with the Communications Research Centre (CRC) to develop a proof-of-concept testbed for a reconfigurable omniband (ROBR) satcom ground terminal using the software radio concept described above. The hardware may consist of general processors, digital signal processor (DSP) boards, application specific integrated circuits (ASICs), or field programmable gate arrays (FPGAs) to perform the processing functions of the particular waveform standard of interest.

One common function that appears in the transmit/receive chain of a ground terminal is filtering. Filter specifications may differ from location to location in the transmit/receive chain. It would be useful to have one software module capable of generating filter coefficients for many types of filters. While many commercial software packages are currently available for filter design, they produce a text file with coefficients that have to be manually integrated into the processing elements of the terminal. Any change in filter specifications would require a new text file to be generated and integrated. The development of a reconfigurable digital filter module for the ROBR project allows the coefficients to be generated or updated while the terminal continues to operate.

## 1.1 Scope

The work documented in this report provides a suite of several kinds of reconfigurable low-pass filters for use in a DSP and more specifically, in the ROBR terminal testbed. The reconfigurable digital filters have been programmed in C. Infinite impulse response (IIR) and finite impulse response (FIR) filters have been implemented for more flexibility. The techniques to design IIR filters are based on conversion of continuous-time IIR systems into discrete-time IIR systems. This project explores and implements two different conversion

methods: the Bilinear Transformation and the Impulse Invariance method. Five major types of IIR filters have been implemented: the Butterworth filter, the Chebyshev filter, the inverse Chebyshev filter, the Elliptical filter and the Bessel filter. In contrast FIR filters are almost entirely restricted to discrete-time implementations. FIR filter design is an approximation of an ideal frequency response using specific approximation methods. Three approaches have been explored and implemented to design optimal equiripple filters. They are the "Frequency Response Sampling Design" method; the "Design by Windowing" method; and the "Parks-McClellan" algorithm. The first two methods consist of sampling the frequency response and performing an inverse discrete Fourier Transform (IDFT) to compute the filter's coefficients. The Parks-McClellan method uses techniques from the approximation theory. The following report provides a description of IIR and FIR systems and filters. The report describes the various filter design methods mentioned above and their implementation for a general purpose processor. The report also describes the adaptation of two of the implemented methods for use on a DSP. A comparison of the generated filter coefficients for each filter with MATLAB implementations is presented. The report also discusses implementation issues related to digital filter design. The reconfigurable digital filter modules are available in the package *digital\_filters*. Further implementation details can be found in the user's guide in the *digital\_filters* package.

## 2 DIGITAL FILTERS

Filters are an important class of systems in signal analysis, signal processing and communication. A filter can be described as a discrete-time system or an analog system that passes certain frequency components while rejecting others. In a more general context, any system that modifies certain frequencies relative to others is also called a filter. The following sections present the fundamental concepts involved in discrete-time or digital filtering.

### 2.1 Discrete-time signal

A discrete-time signal is an indexed sequence of real or complex numbers denoted by  $x[n]$  [1]. It is a function of an integer-valued variable,  $n$ , that represents an instant in time. In practice, discrete-time signals are derived by sampling a continuous-time signal  $x_c(t)$  to produce a sequence of samples. Alternatively, a sequence,  $x[n]$ , can be represented as a sum of scaled, delayed impulses [1] as follows

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \delta[n-k] \quad (1)$$

where

$$x[k] = x_c(kT) \quad (2)$$

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \quad (3)$$

Figure 1 shows an example of a discrete-time signal,  $x[n]$ , representing an arbitrary continuous-time signal,  $x_c(t)$ .

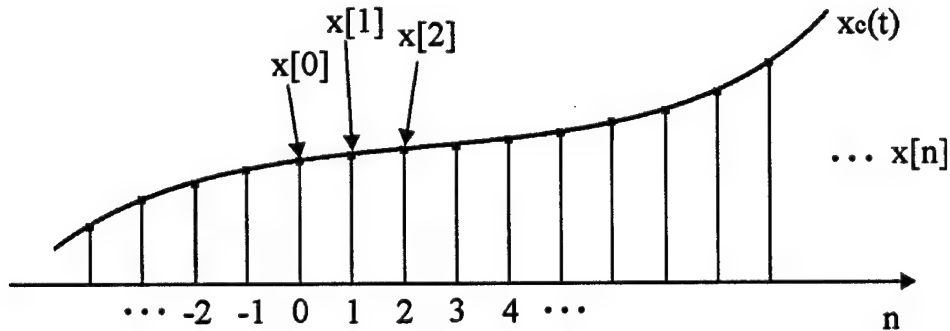
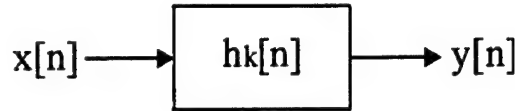


Figure 1 Graphical representation of a discrete-time signal

## 2.2 Linear Time-Invariant (LTI) systems

This section introduces LTI systems, which are very important for understanding discrete filtering. LTI systems may be described in terms of the effect they have on discrete-time signals. Figure 2 shows a block diagram of an LTI system. The input  $x[n]$  and the output  $y[n]$  of the LTI system are discrete-time signals. An LTI system may be viewed as a black box where its output is related to its input by the impulse response (or transfer function)  $h_k[n]$  of the system.



**Figure 2 Representation of a Linear Time-Invariant System**

The impulse response,  $h_k[n]$ , is the response of the system to an impulse  $\delta[n-k]$ . An LTI system can be completely characterized by its impulse response. We can obtain the output of the system from any input by computing

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] h_k[n-k] \quad (4)$$

which is commonly called a *convolution sum* [1] and is denoted by

$$y[n] = x[n] * h_k[n] \quad (5)$$

## 2.3 The Laplace transform and the Z-transform

The Laplace transform is one of the most important tools used in signal analysis. It is used to represent the frequency spectrum of a given signal. In fact, the Fourier transform is a special case of the Laplace transform. The Laplace transform of a function,  $x(t)$ , defined for  $t \in [-\infty, \infty]$ , is given by

$$X(s) = \int_{-\infty}^{\infty} x(t) e^{-st} dt \quad (6)$$

where  $s$  is a complex frequency variable,  $s = \sigma + j\Omega$  [2]. If  $x(t)$  describes the behaviour of a system in the time domain,  $X(s)$  represents the behavior of the same system in the complex frequency domain. It is noted that for the purposes of this report, the symbol  $\Omega$  is

used to represent the continuous frequency variable whereas the symbol  $\omega$  is used to denote the discrete frequency variable as will be described further.

The equivalent transformation for a discrete-time system is the Z-transform. The Z-transform changes the representation of a discrete signal from the time domain to the discrete frequency domain. The Z-transform of a discrete signal  $x[n]$  is given by [2]

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \quad (7)$$

In order to see the relationship between the Laplace and Z- transforms, consider a function,  $x_e(t)$ , which is obtained by sampling a continuous function,  $x_c(t)$ , represented mathematically as

$$x_e(t) = x_c(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) = \sum_{n=-\infty}^{\infty} x_c(nT) \delta(t - nT) \quad (8)$$

where  $T$  is the sampling period. The Laplace transform of  $x_e(t)$  can be written as

$$X_e(s) = \int_{-\infty}^{\infty} x_e(t) e^{-st} dt \quad (9)$$

$$X_e(s) = \sum_{n=-\infty}^{\infty} x_c(nT) e^{-nTs} \quad (10)$$

Using the relationship in Equation (2), the Z-transform,  $X(z)$ , can be compared with the Laplace transform,  $X_e(s)$ , where it can be seen that they are related by a variable change

$$z = e^{Ts} \quad (11)$$

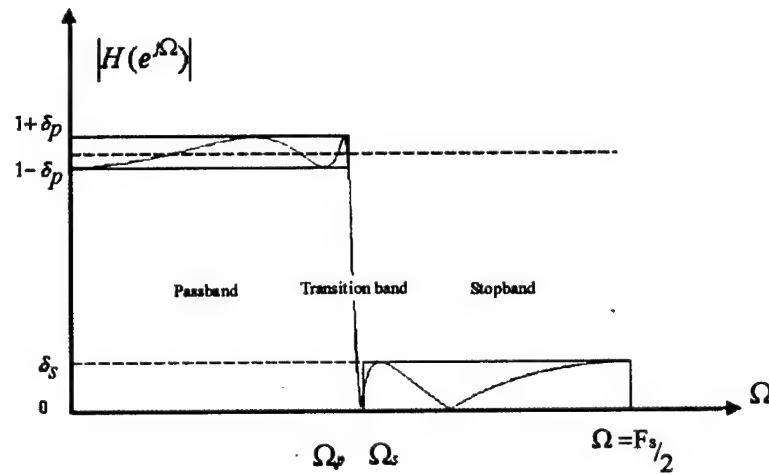
so that

$$X(z) \Big|_{z=e^{Ts}} = \sum_{n=-\infty}^{\infty} x_c(nT) e^{-nTs} = X_e(s) \quad (12)$$

It is noted that the substitution  $z = e^{Ts}$  transforms the  $s = j\Omega$  axis of the complex frequency plane onto a unit circle  $z = e^{jT\Omega}$  [2].

## 2.4 Filter design and module specifications

Filters are an important class of LTI systems. They are used extensively in communications (e.g. low-pass filters). It is convenient to characterize a filter by its frequency response expressed by the magnitude of its transfer function,  $|H(e^{j\Omega})|$ . An example of the magnitude response of the transfer function for a low-pass filter is shown in Figure 3. Parameters that describe the filter characteristics are listed in Table 1.



**Figure 3** Magnitude response for a low-pass filter

Filter order	$n$
Passband frequency	$\Omega_p$
Stopband frequency	$\Omega_s$
Passband ripple	$\delta_p$
Stopband ripple	$\delta_s$
Cutoff frequency	$\Omega_c$
Sampling frequency	$F_s$
Passband attenuation	$A_p$
Stopband attenuation	$A_s$
Transition band	$[\Omega_p, \Omega_s]$

**Table 1** List of specifications for a low-pass filter

Mathematically, the low-pass filter in Figure 3 can be described [1] by

$$1 - \delta_p < |H(e^{j\Omega})| \leq 1 + \delta_p, \quad 0 \leq |\Omega| < \Omega_p \quad (13)$$

$$|H(e^{j\Omega})| \leq \delta, \quad \Omega_s \leq |\Omega| < \frac{F_s}{2} \quad (14)$$

The design process of a filter begins with the filter specifications, which include the constraint on the magnitude of the frequency response, the type of filter and the filter order. Once the specifications have been defined, the next step is to find a set of filter coefficients. The coefficients are simply the values taken from the transfer function  $h_k[n]$  for specific indices that produce the acceptable filter response. After the coefficients have been generated, the next step is to use them to process a signal.

There exist two classes of digital filters: infinite impulse response (IIR) filters and finite impulse response (FIR) filters. They are both described in more detail in the following sections.

### 3 INFINITE IMPULSE RESPONSE FILTERS

#### 3.1 Infinite impulse response (IIR) LTI systems

IIR systems are a subclass of linear time invariant systems and satisfy a linear constant-coefficient difference equation [1] of the form

$$\sum_{k=0}^N a_k y(n-k) = \sum_{k=0}^M b_k x(n-k) \quad a_0 = 1 \quad (15)$$

Applying the Z-transform to both sides of Equation (15) and using the linearity and the time-shifting properties of Z-transforms [1] gives

$$\sum_{k=0}^N a_k z^{-k} Y(z) = \sum_{k=0}^M b_k z^{-k} X(z) \quad a_0 = 1 \quad (16)$$

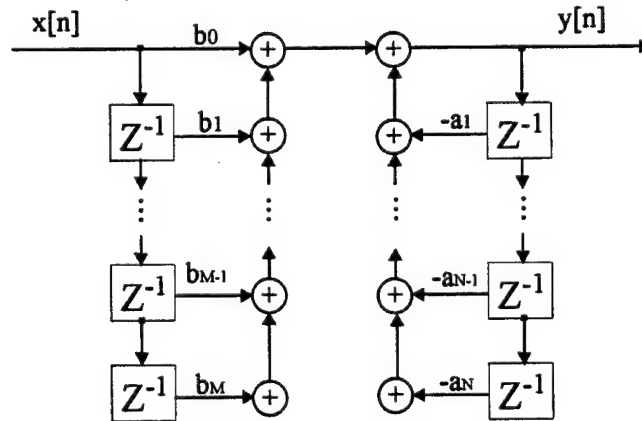
The transfer function in the Z-domain,  $H(z)$ , can now be defined as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^M b_k z^{-k}}{1 + \sum_{k=1}^N a_k z^{-k}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_N z^{-N}} \quad (17)$$

The coefficients,  $b_k$  and  $a_k$ , are the filter coefficients.

##### 3.1.1 Flow diagrams of recursive structures

For implementation on a general or digital processor, IIR systems of the form presented in Equation (15) must be converted into a structure from which an algorithm can be derived. The difference equation given by Equation (15) can be represented graphically as a recursive structure [1]. The IIR structure shown in Figure 4 is referred to as Direct Form I.



**Figure 4** Flow diagram implementing the Direct Form I realization of an IIR filter



Table 2 shows the computational requirements of Direct Form I [3].

Number of multiplications	N + M + 1 per output sample
Number of additions	N + M per output sample
Number of delays	N + M

**Table 2 Computational requirements for Direct Form I structure**

### 3.2 IIR filter design

There are two general approaches used to design IIR filters. The first approach is to design an analog IIR filter and then map it into an equivalent digital filter. This method is computationally efficient and gives a lot of control on the design because the art of analog filter design is highly advanced [1]. The second approach is to use algorithmic and iterative design procedures, which generally requires solving a set of linear or non-linear equations. The first approach has been used in this project.

#### 3.2.1 Analog filter design

##### 3.2.1.1 Classical analog filter approximations

This section presents the characteristics of the five analog filter prototypes that have been used to produce discrete IIR filters. The important characteristics to be considered are the transfer function of the filter, the magnitude and the phase of the frequency response of the filter, the poles and zeroes of the filters. The transfer function of the analog prototypes will be expressed in the continuous frequency domain in terms of its Laplace Transform.

The transfer function of any analog prototype filter may be expressed as

$$H(s) = H_0 \frac{\prod_{l=1}^m s - z_l}{\prod_{k=1}^n s - p_k} \quad (18)$$

where  $z_l$  are the zeroes and  $p_k$  are the poles of the transfer function. The magnitude of the frequency response may be expressed as

$$H(s)H(-s)\Big|_{s=j\Omega} = |H(j\Omega)|^2 \quad (19)$$

The poles of the transfer function [1] are complex numbers and are usually of the form

$$p_k = \sigma_k + j\Omega_k \quad (20)$$

### 3.2.1.2 Butterworth filter properties

A Butterworth filter yields a flat frequency response in the passband and in the stopband as shown in Figure 5. The Butterworth approximation, generally used to design low-pass filters, yields an allpole filter and can be described by the following equations [4][5].

$$\text{normalized transfer function} \quad H(s) = \frac{1}{\prod_{k=1}^n (s - p_k)} = \frac{1}{(s - p_1)(s - p_2) \dots (s - p_n)} \quad (21)$$

$$\text{unnormalized transfer function} \quad H(s) = \frac{\Omega_c^n}{\prod_{k=1}^n (s - p_k)} \quad (22)$$

$$\text{magnitude} \quad |H(j\Omega)|^2 = \frac{1}{1 + \varepsilon^2 \Omega^{2n}} \quad (23)$$

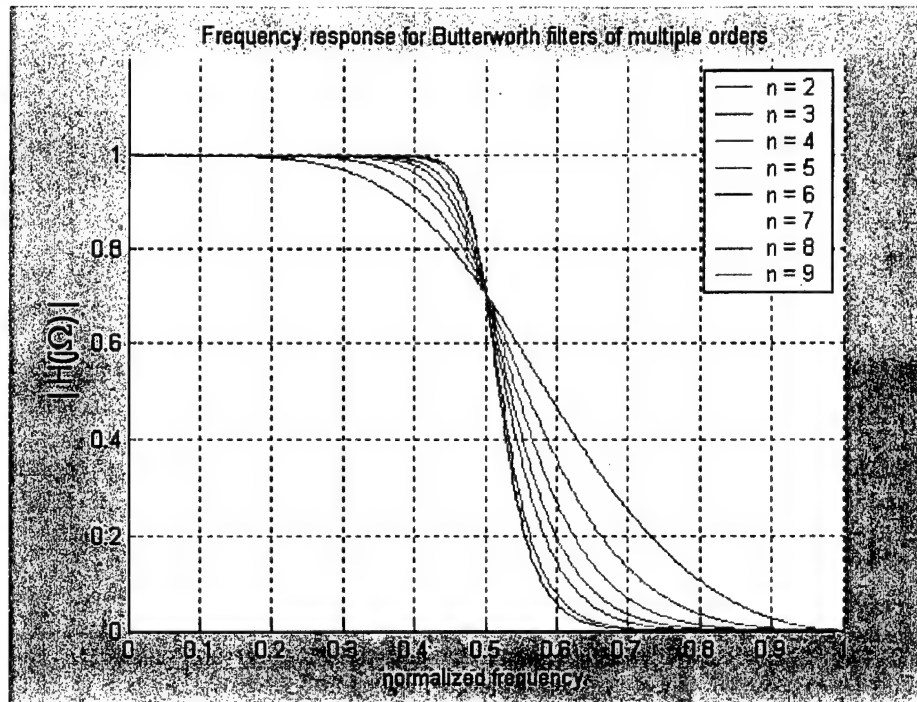
where  $\Omega_c$  = cutoff frequency,  $\varepsilon = \sqrt{10^{\frac{A_p}{10}} - 1}$ , and  $A_p$  = passband attenuation in dB.

The poles of  $H(s)$ ,  $p_k = \sigma_k + j\Omega_k$ , are located at  $2n$  equally spaced points around a circle of radius  $\Omega$  [4][5] and are given by

$$p_k = (-1)^{1/2n} (j\Omega) = \Omega_c \exp \left\{ j \frac{(n+1+2k)\pi}{2n} \right\} \quad k = 0, 1, \dots, n-1 \quad (24)$$

$$\sigma_k = \Omega_c \cos \left( \frac{(n+1+2k)\pi}{2n} \right) \quad (25)$$

$$\Omega_k = \Omega_c \sin \left( \frac{(n+1+2k)\pi}{2n} \right) \quad (26)$$



**Figure 5 Frequency response for a Butterworth filter**

The number of poles equals the order of the analog filter,  $n$ . As shown in Figure 5, the frequency response in the transition band becomes steeper as the filter order increases.

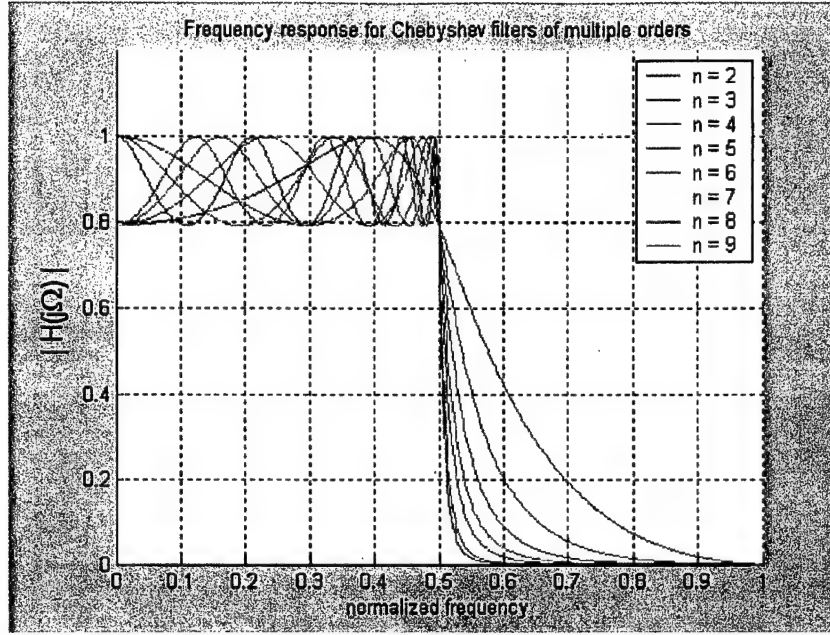
The minimum order that will ensure an attenuation of  $A_s$  or more at frequencies  $\Omega_s$  and above can be obtained [5] by using

$$n = \frac{\log(10^{-A_s/10} - 1)}{2 \log\left(\frac{\Omega_s}{\Omega_c}\right)} \quad (27)$$

where  $\Omega_s$  = stopband frequency

### 3.2.1.3 Chebyshev filter properties

Chebyshev filters are designed to have an amplitude response with relatively sharp transition from the passband to the stopband. This sharpness is achieved at the expense of ripples that are introduced into the response. As with the Butterworth approximation, the Chebyshev approximation yields an allpole filter. Figure 6 shows examples of Chebyshev filters of various order,  $n$ . As the order increases, the ripple in the passband increases. However the tradeoff is a sharper transition from the passband to the stopband.



**Figure 6 Frequency response for a Chebyshev filter**

The transfer function and magnitude response of a Chebyshev filter [4][5] is given by

Transfer function 
$$H(s) = H_0 \prod_{k=0}^{n-1} \frac{-p_k}{(s - p_k)} \quad (28)$$

Magnitude response 
$$|H(j\Omega)|^2 = \frac{1}{1 + \varepsilon^2 T_n^2(\Omega)} \quad (29)$$

where the static gain,  $H_0$ , is given by

$$H_0 = \begin{cases} \prod_{k=1}^n (-p_k) & \mathbf{n \text{ odd}} \\ 10^{r_p/20} \prod_{k=1}^n (-p_k) & \mathbf{n \text{ even}} \end{cases} \quad (30)$$

The parameter,  $\varepsilon$ , is dependent on the passband ripple,  $r_p = 20 \log(\delta_p)$ , as follows

$$\varepsilon = \sqrt{10^{r_p/10} - 1} \quad (31)$$

and  $T_n(\Omega)$  is the Chebyshev polynomial given by

$$T_n(\Omega) = \begin{cases} \cos(n \cos^{-1}(\Omega)) & 0 \leq \Omega \leq 1 \\ \cosh(n \cosh^{-1}(\Omega)) & \Omega \geq 1 \end{cases} \quad (32)$$

The  $2n$  poles of a Chebyshev filter [4][5] are given by Equation (29) where

$$\sigma_k = \Omega_c \cos \left[ \frac{(2k-1)}{2n} + \frac{\pi}{2} \right] \left[ \frac{(1/\gamma) - \gamma}{2} \right] \quad k = 0, 1, \dots, n-1 \quad (33)$$

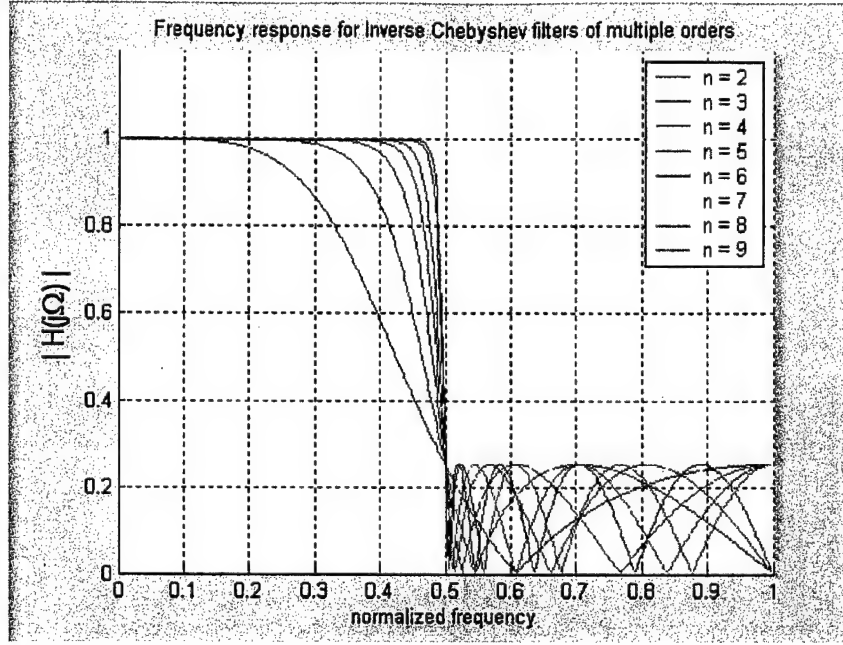
$$\Omega_k = \Omega_c \sin \left[ \frac{(2k-1)}{2n} + \frac{\pi}{2} \right] \left[ \frac{(1/\gamma) + \gamma}{2} \right] \quad k = 0, 1, \dots, n-1 \quad (34)$$

and

$$\gamma = \left( \frac{1 + \sqrt{1 + \varepsilon^2}}{\varepsilon} \right)^{1/n} \quad (35)$$

#### 3.2.1.4 Inverse Chebyshev properties

The inverse Chebyshev approximation yields a filter which has zeroes and poles. Depending on whether the order is even or odd, the filter will have as many zeroes as poles, or  $n-1$  zeroes and  $n$  poles. Figure 7 shows the frequency response of an inverse Chebyshev filter for different orders. Just as the Butterworth and Chebyshev filters showed, the frequency response becomes steeper in the transition band as the order of the filter increases. However, the inverse Chebyshev filter frequency response exhibits a flat response in the passband and ripples in the stopband. The ripples in the stopband increase as the filter order increases.



**Figure 7 Frequency response for Inverse Chebyshev filters**

The transfer function of an Inverse Chebyshev filter [4] is given by

$$H(s) = \prod_{k=0}^{n-1} \frac{a_k}{b_k} \frac{s - b_k}{(s - a_k)} \quad a_k = \frac{\Omega_s^2}{s_k} \quad (36)$$

and its magnitude response is

$$|H(j\Omega)|^2 = \frac{1}{1 + \varepsilon^2 [T_n^2(\Omega)]^{-1}} \quad (37)$$

where  $T_n(\Omega)$  is again the Chebyshev polynomial given by Equation (32), and where  $\varepsilon$  depends on the stopband ripple in dB,  $r_s = 20 \log(\delta_s)$  as follows,

$$\varepsilon = \sqrt{10^{r_s/10} - 1} \quad (38)$$

The poles of an Inverse Chebyshev filter [4] are given by Equation (36) where

$$\sigma_k = \left( \Omega_c \cos \left[ \frac{(2i+1)\pi}{2n} + \frac{\pi}{2} \right] \sinh \mu \right)^{-1} \quad i = 0, 1, \dots, n-1 \quad (39)$$

$$\Omega_k = \left( \Omega_c \sin \left[ \frac{(2i+1)\pi}{2n} + \frac{\pi}{2} \right] \cosh \mu \right)^{-1} \quad i = 0, 1, \dots, n-1 \quad (40)$$

where

$$\mu = \frac{\sinh^{-1}(1/\epsilon^{-1})}{n} \quad (41)$$

The zeroes of an Inverse Chebyshev filter [4] are given by

For n odd

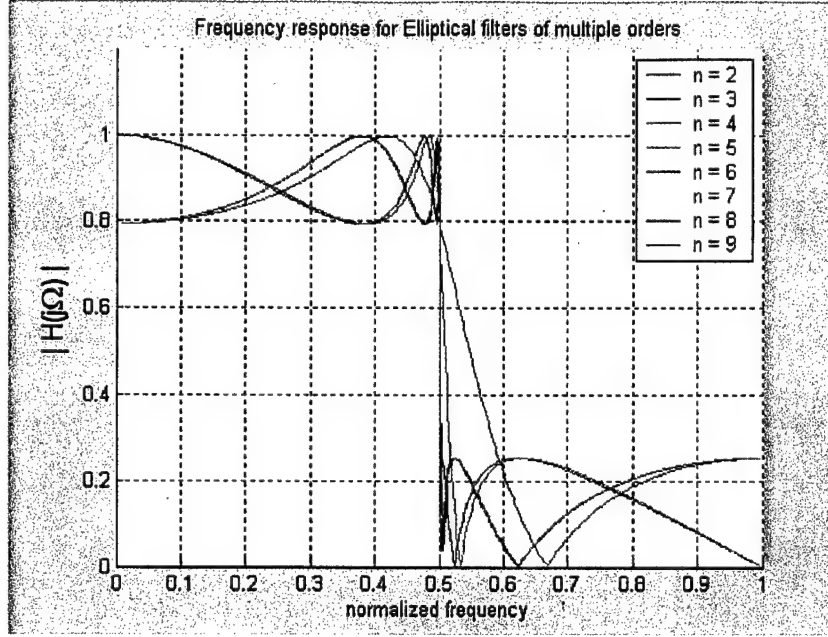
$$z_k = \frac{j}{\cos \left( \frac{(2k+1)\pi}{2n} \right)} \quad \begin{array}{l} k = 0, 1, \dots, n-1 \\ k \neq n \end{array} \quad (42)$$

For n even

$$z_k = \frac{j}{\cos \left( \frac{(2k+1)\pi}{2n} \right)} \quad k = 0, 1, \dots, n-1 \quad (43)$$

### 3.2.1.5 Elliptical filter properties

By allowing ripples in the passband, Chebyshev filters obtain better frequency selectivity than Butterworth filters because of the sharper transition band. Elliptical filters improve upon the performance by permitting ripples in both the passband and stopband. Figure 8 shows the frequency response of an Elliptical filter for different values of filter order.



**Figure 8 Frequency response for an Elliptical filter**

The transfer function of an Elliptical filter [4][5] is given by

$$H(s) = \frac{H_0}{D_0(s)} \prod_{i=1}^l \frac{s^2 + a_{0i}}{s^2 + b_{1i}s + b_{0i}} \quad (44)$$

where

$$l = \begin{cases} \frac{n-1}{2} & \text{for } n \text{ odd} \\ \frac{n}{2} & \text{for } n \text{ even} \end{cases} \quad (45)$$

$$D_0(s) = \begin{cases} s + \sigma_0 & \text{for } n \text{ odd} \\ 1 & \text{for } n \text{ even} \end{cases} \quad (46)$$

and  $\sigma_0$  is a constant defined in Appendix B.

The analog static gain,  $H_0$ , can be calculated [4] using



$$H_0 = \begin{cases} \sigma_0 \prod_{i=1}^l \frac{b_{0i}}{a_{0i}} & \text{for } n \text{ odd} \\ 10^{-0.05 A_p} \prod_{i=1}^l \frac{b_{0i}}{a_{0i}} & \text{for } n \text{ even} \end{cases} \quad (47)$$

and the magnitude response [4][5] using

$$|H_c(j\Omega)|^2 = \frac{1}{1 + \varepsilon^2 R_n^2(\Omega)} \quad (48)$$

$R_n(\Omega)$  is the Chebyshev Rational Function with respect to the centre frequency  $\sqrt{\Omega_p \Omega_s} = \Omega_0 = 1$  and where  $\varepsilon$  is the ripple factor which depends on the passband ripple,  $\delta_p$ , or on the stopband ripple in decibels,  $r_s$  [3] as follows

$$\varepsilon = \sqrt{\frac{2\delta_p - \delta_p^2}{1 - 2\delta_p + \delta_p^2}} = \sqrt{10^{r_s/10} - 1} \quad (49)$$

The Rational Normalized Function  $R_n(\Omega)$  is given by [2]

$$R_n(\Omega) = \begin{cases} \Omega \prod_{i=1}^{(n-1)/2} \frac{\Omega_i^2 - \Omega^2}{1 - \Omega_i^2 \Omega^2} & \text{for } n \text{ odd} \\ \prod_{i=1}^{n/2} \frac{\Omega_i^2 - \Omega^2}{1 - \Omega_i^2 \Omega^2} & \text{for } n \text{ even} \end{cases} \quad (50)$$

The Elliptical filter poles can be computed following a calculation algorithm. Steps to calculate the values  $a_{0i}$ ,  $b_{0i}$  and  $b_{li}$  of the transfer function are taken from [4] and are listed in Appendix B. Once these values are calculated it is easy to obtain the poles, the zeroes and the gain of the analog filter which are then used to compute the coefficients of the filter [4]. The important parameters that have to be considered for the design of an analog Elliptical filter are

$$\begin{aligned} \Omega_p &= \text{passband frequency} \\ \Omega_s &= \text{stopband frequency} \\ A_p &= \text{maximum passband loss (dB)} \\ A_s &= \text{maximum stopband loss (dB)} \\ k &= \text{selectivity factor} = \Omega_p / \Omega_s \end{aligned}$$

Using the quadratic formula, the  $i^{\text{th}}$  pair of complex pole values [4] can be expressed as

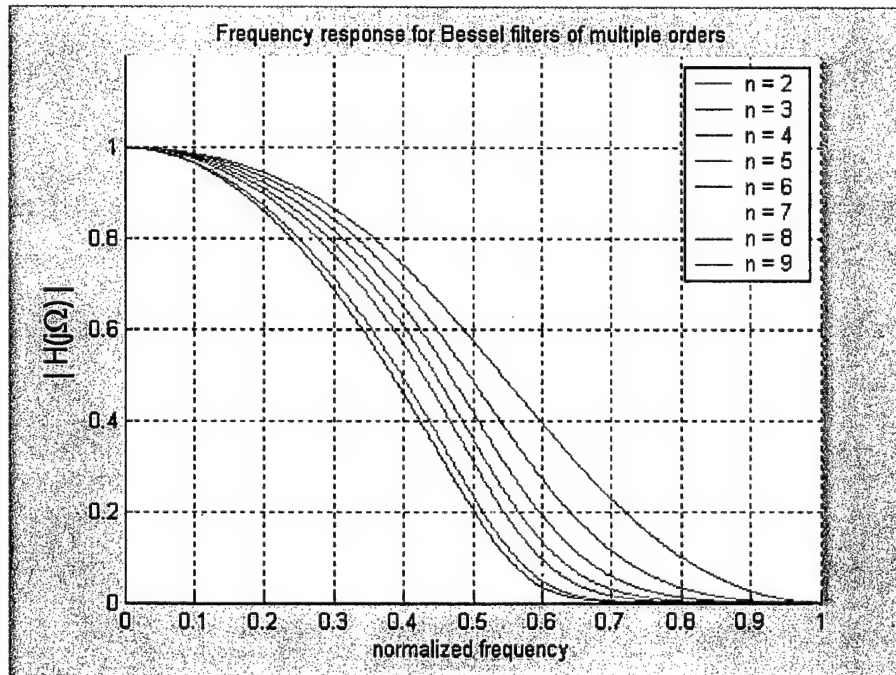
$$p_i = \frac{-b_{li} \pm \sqrt{b_{li}^2 - 4b_{0i}}}{2} \quad i = 1, 2, \dots, l \quad (51)$$

The zeroes occur at

$$z_i = \pm j\sqrt{a_{0i}} \quad i = 1, 2, \dots, l \quad (52)$$

### 3.2.1.6 Bessel filter properties

Bessel filters are designed to have maximally flat group-delay characteristics. As a consequence, there is no ripple in the impulse and step response. On the other hand, the rolloff of the frequency response is more gradual, making the transition band wider than for Butterworth or Chebyshev type filters. The Bessel filter is also an allpole filter. Figure 9 shows the frequency response of Bessel filters for various values of  $n$ .



**Figure 9** Frequency response for a Bessel filter for different values of filter order,  $n$

The transfer function of a Bessel filter [4] is expressed as follows

$$H(s) = \frac{b_0}{q_n(s)} \quad (53)$$

where

$$q_n(s) = \sum_{k=1}^n b_k s^k \quad (54)$$

$$b_k = \frac{2(n-k)!}{2^{n-k} k! (n-k)!} \quad (55)$$

The following recursion formula [5] is used to determine  $q_n(s)$  from  $q_{n-1}(s)$  and  $q_{n-2}(s)$

$$q_n = (2n-1)q_{n-1} + s^2 q_{n-2} \quad (56)$$

Using this recursion formula involves the computation of the poles of the analog transfer function with a numerical analysis formula. These poles are the roots of Equation (53). Using these formulas involves more complexity in the algorithm and there is no guarantee that the computation of the roots will converge. A very efficient way to implement the Bessel filter is by using pre-computed values for its poles. The Bessel filter was implemented using a look up table for filter orders of up to  $n = 25$ . Poles of the analog transfer function of the Bessel filter have been computed using the *Signal Processing* toolbox from MATLAB and are used in this project. The analog poles are converted to discrete values using one of the analog-to-digital conversion methods shown in the next section to obtain a discrete design for the implementation.

### 3.2.2 Conversion of analog IIR filters for digital implementation

This section describes two analog-to-digital conversion methods that have been used for IIR filter implementation in this project. The idea of the conversion methods is to transform the analog transfer function expressed in the analog frequency domain or the s-plane into a discrete transfer function expressed in the discrete frequency domain or the z-plane.

#### 3.2.2.1 Bilinear transformation

The bilinear transform [1] is a mapping from the s-plane to the z-plane defined by

$$H(z) = H_c \left[ \frac{2}{T} \left( \frac{1-z^{-1}}{1+z^{-1}} \right) \right] \quad (57)$$

where  $H_c(s)$  is the Laplace transform of a continuous function.

In the previous sections, the equations for the analog poles of the transfer functions of different IIR filters expressed in the S-domain were presented. To compute the discrete poles of the digital IIR filter from the analog poles of the analog prototype the following relationship is used.

$$s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \quad (58)$$

or conversely,

$$z = \frac{2 - Ts}{2 + Ts} \quad (59)$$

where  $T$  is the sampling period.

### 3.2.2.1.1 Frequency warping function

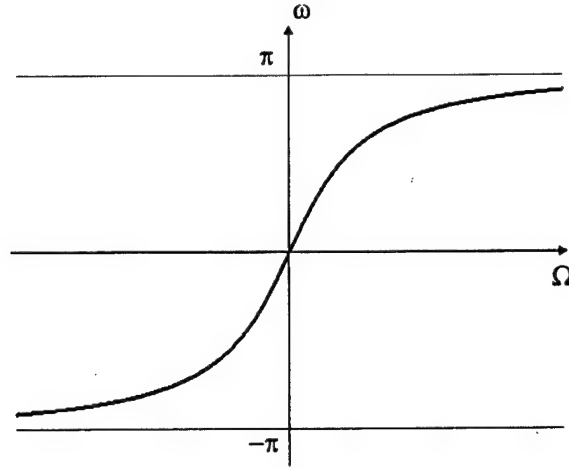
In bilinear transformation, the relationship between the analog or continuous frequency,  $\Omega$ , and the digital frequency,  $\omega$ , in the Z-domain [1] is given by

$$\Omega = \frac{2}{T} \tan\left(\frac{\omega}{2}\right) \quad (60)$$

Conversely, to compute the digital frequency corresponding to an analog frequency in the S-domain, the following equation applies

$$\omega = 2 \arctan\left(\frac{\Omega T}{2}\right) \quad (61)$$

Equation (60) is plotted and shown in Figure 10. It is noted that the range of analog frequencies  $-\infty \leq \Omega \leq \infty$  maps to normalized digital frequencies  $-\pi \leq \omega \leq \pi$ . The frequency  $\omega = \pi$  corresponds to half the normalized sampling frequency  $F_s/2 = 2\pi$ . If viewed from the point of view of Nyquist's Theorem, since the bilinear transformation is able to map all continuous frequencies,  $\Omega$ , to digital frequencies below  $F_s/2$ , there are no aliased components resulting from the conversion of an analog filter to a digital one. However, the effect of this transformation is the non-linear compression of the discrete frequency axis as shown in Figure 10.



**Figure 10 Mapping of continuous frequency to digital frequency**

### 3.2.2.2 Impulse invariance method

The “Impulse Invariance” method consists of sampling the impulse response of a continuous-time system,  $h_c(t)$  to get a discrete impulse response,  $h[n]$ , as follows

$$h[n] = T h_c(nT) \quad (62)$$

From Nyquist’s sampling theory, it can be shown [1] that the frequency response of the discrete-time filter is related to the frequency response of the continuous-time filter by [1]

$$H(e^{j\omega}) = \sum_{k=-\infty}^{\infty} H_c\left(j\frac{\omega}{T} + j\frac{2\pi}{T}k\right) \quad (63)$$

However, in the case of an ideal bandlimited continuous-time filter,

$$H_c(j\Omega) = 0, \quad |\Omega| \geq \pi/T \quad (64)$$

which reduces Equation (63) to

$$H(e^{j\omega}) = H_c\left(j\frac{\omega}{T}\right) \quad |\omega| \leq \pi \quad (65)$$

It follows from Equation (64) that the discrete-time and continuous-time frequencies are related linearly by

$$\omega = \Omega T \quad (66)$$

In practice, the frequency response of a continuous-time filter is not bandlimited as described in Equation (65). As a result, aliasing can occur between successive terms of Equation (63). The presence of aliasing has an effect on the design of a discrete-time filter using the Impulse Invariance method. However, if the frequency response of the continuous-time filter at higher frequencies is sufficiently low and the sampling frequency is sufficiently high enough, then aliasing is minimized.

It is shown in [1] that the transformation from continuous-time to discrete-time can be achieved by considering the continuous-time frequency response as a partial fraction so that

$$H_c(j\Omega) = H_c(s) = \sum_{k=1}^N \frac{A_k}{s - s_k} \quad (67)$$

$A_k$  may be obtained [5] by computing,

$$A_k = [(s - p_k)H_c(s)] \Big|_{s=p_k} \quad (68)$$

Taking the inverse Laplace transform, the impulse response of the continuous-time filter becomes

$$h_c(t) = \begin{cases} \sum_{k=1}^N A_k e^{s_k t} & t \geq 0 \\ 0 & t < 0 \end{cases} \quad (69)$$

The impulse response of the discrete-time filter, obtained by sampling  $h_c(t)$  from Equation (62) becomes

$$h[n] = Th_c(nT) = T \sum_{k=1}^N A_k (e^{s_k T})^n u[n] \quad (70)$$

where  $u[n]$  is the unit step function.

By taking the Z-transform of Equation (70), the discrete transfer function of the filter is therefore given by

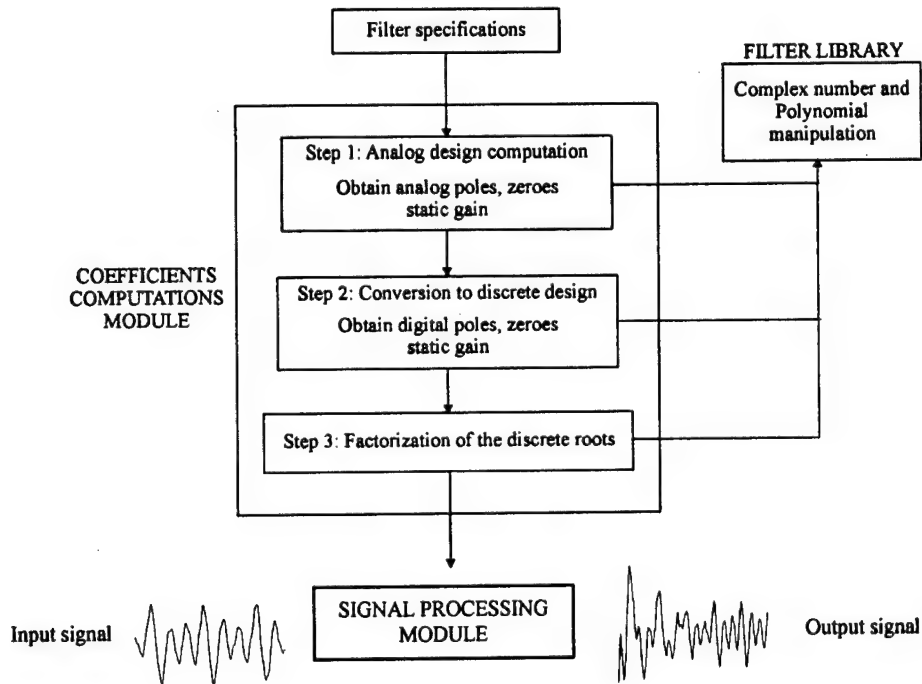
$$H(z) = \sum_{k=1}^N \frac{A_k}{1 - e^{(s_k T)} z^{-1}} \quad (71)$$

### 3.3 IIR filtering module implementation

#### 3.3.1 System

The IIR filters implemented consist of two important components, the coefficients computation module and the signal processing module. The coefficients computation module is responsible for designing the filter and computing the coefficients. In order to verify the coefficients computation module, a signal processing module is implemented that uses the coefficients generated by the computation module to filter a test signal.

The coefficients computation module operates in three basic steps as shown in Figure 11. First, it computes the analog poles, zeroes and static gain of the analog design. Secondly, it transforms the analog filter design to a digital design and obtains the digital poles, zeroes and static gain. The final step consists of factorizing the digital poles and zeroes to obtain the filter coefficients. Two analog-to-digital conversion methods have been implemented and are compared.



**Figure 11 Block diagram of the implementation of the reconfigurable IIR filtering modules**

Each of the steps outlined above makes use of a header file called "Filter.h". The header file contains routines for complex number manipulation and polynomial manipulation and is listed in Appendix C. Complex number manipulations include computing the norm, conversion from Euler representation to trigonometric representation using the formula,  $re^{j\omega} = \cos \omega + j \sin \omega$ , multiplication and division of two complex numbers, computing the

square root of a complex number, and computing a number raised to a complex power,  $n^{x+jy}$ . Polynomial manipulations include the multiplication of two polynomials and computing an expansion formula to obtain the roots of a polynomial.

The following two sections describe the coefficients computation and signal processing modules in more detail. The software developed for this project is assembled in a *digital\_filters* package for future use in the ROBR.

### 3.3.2 IIR coefficients computation modules

The principle function of these modules is to compute the constants  $a_k$  and  $b_k$  for an IIR filter given by Equation (17). Five IIR coefficients computation modules have been implemented based on the analog designs presented in Section 3.2. Table 3 lists the implemented modules.

Filter type	Module name
Butterworth filter	butter
Chebyshev filter	cheb1
Inverse Chebyshev filter	cheb2
Elliptical filter	Ellip
Bessel filter	Bessel

**Table 3 IIR Filter types implemented by the corresponding modules**

More information about how to use each module can be found in a user's guide included with the software developed. All modules require two input parameters, the digital cutoff frequency in Hertz and the sampling frequency also in Hertz. To enable the analog design computation block to calculate the analog values of the analog design the input cutoff frequency needs to be normalized and mapped as in Equation (60). Figure 12 shows the implementation of these operations.

```
// normalization of the digital cutoff frequency over 0 to 2π
float Wd = fc * PI / (Fs / 2.0);
// computation of the analog frequencies
float Wc = 2.0/(1.0/Fs) * tan(Wd/2.0);
```

**Figure 12 Cutoff frequency normalization and analog frequency computation**

$Wd$  is the normalized digital cutoff frequency received by the modules.  $Wc$  is the analog cutoff frequency required to compute the analog design, and  $F_s$  is the sampling frequency.



### 3.3.2.1 Analog design computation

The first step of the reconfigurable IIR module is to compute the analog poles, zeroes and static gain for a desired analog filter design. Figure 13 shows the implementation of the analog design computation block for the Chebyshev module (cheb1). This block implements the equations shown in Section 3.2.1.3. The variables,  $p$ ,  $z$  and  $K$  are the arrays that contain the poles, the zeroes and the analog static gain, respectively. The poles are computed using Equations (33) and (34) and the analog static gain  $K$  (introduced as  $H_0$  in Section 3.2.1) is obtained by computing Equation (30). Recall from Section 3.2.1 that the Chebyshev filter is an allpole filter and thus, does not have zeroes. This is why the elements of array  $z$  are set to zero. The same procedure is carried out for the other IIR filters with the appropriate equations for poles, zeroes, and static gain.

As explain earlier in Section 3.2.1.6, the Bessel filter is implemented using a lookup table. No zeroes are computed because this filter is an allpole filter. Figure 14 shows the implementation of the Bessel coefficients as a look up table. The module considers filters up to order 25. The poles stored in the table have been calculated with MATLAB's *besselap* function from the *Signal Processing* toolbox.

```
//computation of the analog poles
for (i=1;i<=N;i++)
{
    z[i-1].Q = 0;
    z[i-1].I = 0;
    p[i-1].Q = sin((2*i-1)*PI/(2*N))*((1.0/gamma)-gamma)/2.0;
//formula for analog poles
    p[i-1].I = cos((2*i-1)*PI/(2*N))*((1.0/gamma)+gamma)/2.0;
    p[i-1] = set_to_zero(p[i-1]);
    K = multc(K.Q,K.I,-1*p[i-1].Q,-1*p[i-1].I);
//computation of the analog static gain
}

//if analog static gain the order is even, adjust the analog gain
if(N%2==0)
    K.Q /=sqrt((1.0 + epsilon*epsilon));
```

**Figure 13 Chebyshev analog design computation algorithm**

[illegible]

**Figure 14 Implementation of the Bessel analog design computation block as a lookup table**

### 3.3.2.2 Analog-to-digital Conversion

The analog-to-digital conversion block is the 2<sup>nd</sup> step in the coefficients computation module. Figure 15 shows the implementation of the conversion block code based on the Bilinear Transform method. This portion of code belongs to the function “*void bilinear()*”, which can be found in the file “*bilinear.c*” in the *digital\_filters* package. The bilinear conversion function takes the analog poles, zeroes and static gain computed in the previous step and computes the corresponding digital poles, zeroes and static gain.

### 3.3.2.3 Factorization

Factorization of the zeroes and poles of the discrete transfer function obtained from the analog design is done using an expansion formula. Using the discrete-time transfer function given in Equation (17), the poles and zeroes of the IIR filter are the roots of the denominator and numerator polynomials respectively. The expansion formula may be expressed as follows

$$c_{n+1} = c_{n+1} - c_n \cdot e_m \quad (72)$$

where  $c_n$  are the computed coefficients and  $e_m$  are the roots of the polynomial. The variable  $n$  is the order of the polynomial and is decremented by one at each iteration. The variable  $m$  equals the order of the polynomial (i.e.  $n = m$  for the first iteration) and stays constant for a set of iterations. The factorization will yield  $n+1$  coefficients where  $n$  is the filter order. Figure 16 shows the implementation of the expansion formula. This portion of code belongs to the function “*void coeff()*” included in the header file “Filter.h”.

```

// discretization of the zeroes

for(i = 0;i < M;i++)
    zd[i]=z[i];

for(i = M;i < N;i++)
{
    zd[i].Q=-1.0;
    zd[i].I=0.0;
}

for(i = 0;i < M; i++)
{
    zd[i] = divc(2.0 + (zd[i].Q * Wc/Fs ),zd[i].I * Wc/Fs ,2.0 - zd[i].Q
                * Wc/Fs ,-1.0 * zd[i].I * Wc/Fs);
    zd[i] = set_to_zero(zd[i]);
    tempNKd = multc(tempNKd.Q,tempNKd.I,(2.0*Fs/(Wc)-z[i].Q),
                    (-1.0*z[i].I));
    tempNKd = set_to_zero(tempNKd);
}

// discretization of the poles

for(i = 0;i < N; i++)
{
    pd[i] = divc(2.0 + (p[i].Q * Wc/Fs ),p[i].I * Wc/Fs ,
                2.0 - p[i].Q * Wc/Fs ,-1.0 * p[i].I * Wc/Fs);
    pd[i] = set_to_zero(pd[i]);
    tempDKd = multc(tempDKd.Q,tempDKd.I,(2*Fs/(Wc)-p[i].Q),
                    (-1.0*(p[i].I)));
}

tempKd = divc(tempNKd.Q,tempNKd.I,tempDKd.Q,tempDKd.I);
Kd = (multc(tempKd.Q,tempKd.I,K.Q,K.I)).Q;

```

**Figure 15 Bilinear transform analog-to-digital conversion algorithm**

```

num = multc(c[n].Q,c[n].I,e[m].Q,e[m].I);

c[n+1].Q = c[n+1].Q - num.Q;

c[n+1].I = c[n+1].I - num.I;

if(n > 0)
    return coeff(&c[0],e,n-1,m);

else
    return 0;

```

**Figure 16 Expansion formula algorithm**

The function “*void coeff()*” is called by the coefficients computation modules after the analog-to-digital conversion step. The function will fill the empty array *cD[n+1]*, where *n* is the order of the IIR filter with the desired filter’s coefficients. More information about this subroutine is included in the user’s guide.

### 3.3.3 IIR signal processing module

The IIR signal processing module has been implemented based on the Direct Form I implementation of the difference equation presented in Section 3.1. This signal processing module can be found in the package *digital\_filter* under the name of *IIRDFI*.

The signal processing module computes the output of a system using the IIR filter coefficients generated by the coefficients computation module. The input test signal used was an impulse sequence.

Figure 17 shows the implementation of the IIR signal processing algorithm. Details on how to execute the IIR filter modules and signal processing module are described in Section 4.3.4.

```

for(i=0;i<numcoef;i++)
    coeffden[i] = divc(a0.Q,a0.I,coeffden[i].Q,coeffden[i].I);

for(i=0;i<numsamples;i++)
{
    sumnum.Q=0.0;
    sumnum.I=0.0;
    sumden.Q=0.0;
    sumden.I=0.0;

    for(j=0;j<numcoef;j++)
    {
        indx = i - j;
        if(indx < 0) break;
        else
        {
            sumnum.Q += coeffnum[j].Q * samples[indx];
            sumnum.I += coeffnum[j].I * samples[indx];
        }
    }

    for(j=1;j<numcoef;j++)
    {
        indx = i - j;
        if(indx < 0) break;
        else
        {
            sumden.Q+=multc(coeffden[j].Q,coeffden[j].I,
                            output[j-1].Q,output[j-1].I).Q;
            sumden.I+=multc(coeffden[j].Q,coeffden[j].I,
                            output[j-1].Q,output[j-1].I).I;
        }
    }

    for(j=numcoef-1;j>0;j--)
        output[j]=output[j-1];

    output[0].Q=sumnum.Q-sumden.Q;
    output[0].I=sumnum.I-sumden.I;
}

```

**Figure 17 IIR signal processing module algorithm**

## 4 FINITE IMPULSE RESPONSE FILTERS

### 4.1 FIR LTI systems

The FIR signal processing module implements an FIR structure called the Direct Form to process digital signals. For FIR systems, the transfer function  $H(z)$  has no poles except at  $z = 0$ . Thus,  $H(z)$  is simply a polynomial in  $z^{-1}$  of the form

$$H(z) = \sum_{k=0}^M h_k z^{-k} = h_0 + h_1 z^{-1} + h_2 z^{-2} + \dots + h_M z^{-M} \quad (73)$$

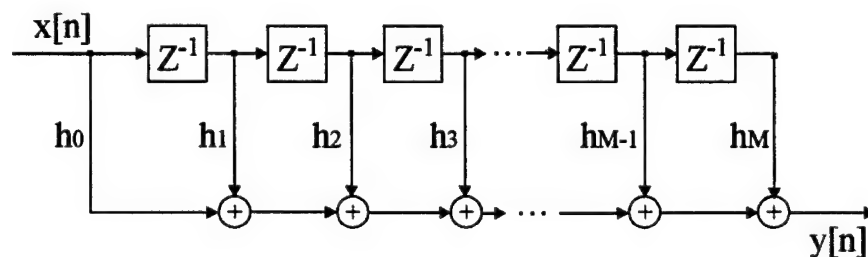
The output of such a filter is

$$\begin{aligned} y(n) &= \sum_{k=0}^M h(k)x(n-k) \\ &= h_0 + h_1 x(n-1) + h_2 x(n-2) \dots h_M x(n-M) \end{aligned} \quad (74)$$

which is the computational sum introduced in Section 2.2.

#### 4.1.1 Flow diagrams for non-recursive structures

The flow diagram shown in Figure 18 illustrates the convolution sum that relates an FIR filter's output to its input. This structure is called the Direct Form. Unit delays are denoted by  $z^{-1}$  as shown in the figure below



**Figure 18 Flow diagram of the Direct Form realization of an FIR filter**

Table 4 shows the computational requirements of Direct Form implementation of an FIR filter [3].

Number of multiplications	$N + 1$ per output sample
Number of additions	$N$ per output sample
Number of delays	$N$

**Table 4 Computational requirements of the Direct Form I structure of FIR filters**

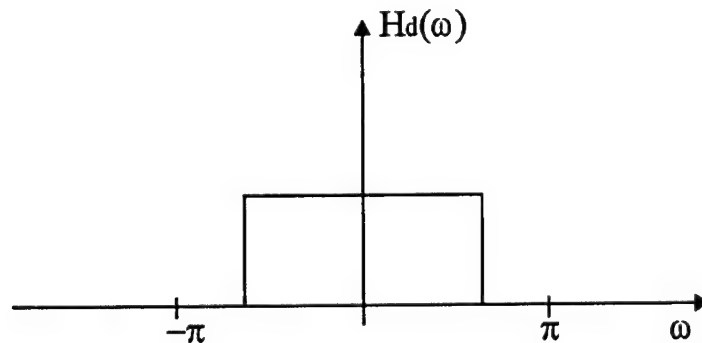
## 4.2 FIR filter design

Recall for IIR filters that the design techniques were based on transformation of continuous-time IIR systems into discrete-time systems. In contrast, FIR filter design is almost entirely restricted to discrete-time implementation. Consequently, the design techniques for FIR filters are based on directly approximating the desired frequency response of the discrete-time system. Furthermore, most of these approximation techniques avoid the problem of factorization that complicates the design of IIR filters. In the context of this project, three techniques have been implemented to compute the FIR filter coefficients of interest: the “Frequency Sampling Design” technique; the “Design by Windowing” method; and the “Parks-McClellan” method. The following sections introduce these three methods.

In addition, the design of a Gaussian filter and a digital integrator is included in this section. A Gaussian filter and a digital integrator are used in the premodulation stage of a Gaussian minimum shift keying (GMSK) modulator that is to be implemented in the ROBR.

### 4.2.1 “Frequency Sampling Design” filter module

The “Frequency Sampling Design” method is a straightforward design procedure. The frequency response of an ideal filter is sampled and each sample of the frequency response is a coefficient. Figure 19 shows the frequency response of an ideal low-pass filter. To get the values of the coefficients in the time domain, an IDFT is performed on the samples collected [1][3][5].



**Figure 19 The desired frequency response  $H_d(\omega)$  of an ideal low-pass filter**

The desired frequency response is uniformly sampled at N equally spaced points between 0 and  $2\pi$  to yield

$$H(k) = H_d \left( e^{j \frac{2\pi k}{N}} \right) \quad k = 0, 1, \dots, N-1 \quad (75)$$

These samples constitute an N-point DFT, whose inverse is the impulse response of an FIR filter of order N-1:

$$h[n] = \frac{1}{N} \sum_{k=0}^{N-1} H(k) e^{j 2\pi k n / N} \quad (76)$$

The inverse DFT can be modified to take advantage of symmetry conditions. Table 5 shows adapted IDFT formulas to the four types of FIR filters. [5]

Type		$h[n]$ for $n = 0, 1, 2, \dots, N-1$
1	$h[n]$ symmetric N odd	$\frac{1}{N} \left\{ H(0) + \sum_{k=1}^M 2H(k) \cos \left[ \frac{2\pi(n-M)k}{N} \right] \right\}$
2	$h[n]$ symmetric N even	$\frac{1}{N} \left\{ H(0) + \sum_{k=1}^{(N/2)-1} 2H(k) \cos \left[ \frac{2\pi(n-M)k}{N} \right] \right\}$
3	$h[n]$ asymmetric N odd	$\frac{1}{N} \left\{ \sum_{k=1}^M 2H(k) \sin \left[ \frac{2\pi(M-n)k}{N} \right] \right\}$
4	$h[n]$ asymmetric N even	$\frac{1}{N} \left\{ H\left(\frac{N}{2}\right) \sin[\pi(M-n)] + \sum_{k=1}^{(N/2)-1} 2H(k) \sin \left[ \frac{2\pi(n-M)k}{N} \right] \right\}$

**Table 5 Inverse Discrete Fourier Transform formulas for FIR Design**

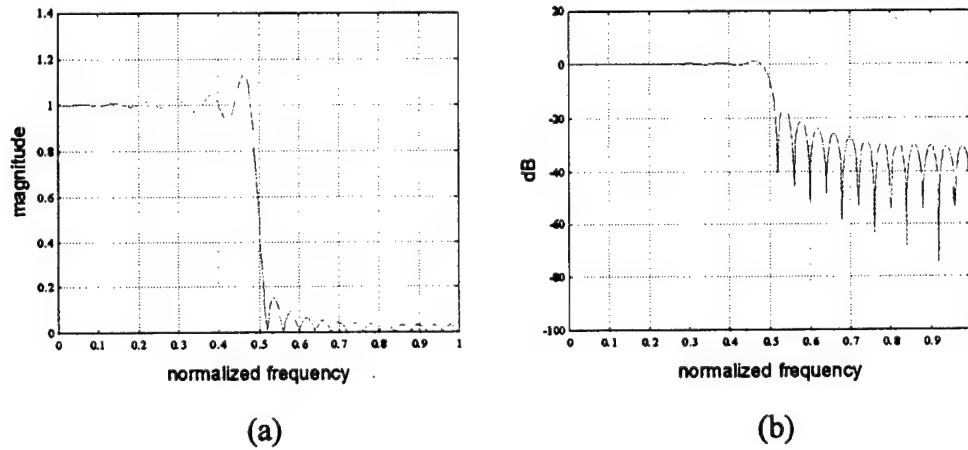
In this project, the first type and the second type of FIR filter have been implemented for a low-pass filter with symmetric impulse response.

#### 4.2.1.1 Gibbs phenomenon

One problem related to the approximation methods used to produce FIR filter coefficients is Gibbs phenomenon. Figure 20 shows the amplitude of the frequency response and the power



spectrum of a low-pass FIR filter affected by Gibbs phenomenon. The coefficients of the filter were computed with the “Frequency Sampling Design” method. The frequency response has an oscillating behaviour that is more pronounced near the edge of the passband. This behavior is known as Gibbs phenomenon and is the result of approximating a discontinuity in the desired frequency response. In [3] it is noted that if a function with a discontinuity is approximated by a Fourier series, there is an overshoot in the region near the discontinuity. As the number of Fourier series terms increases, the squared error decreases and approaches zero as the number of terms approaches infinity. However, the maximum value of the overshoot, and therefore the maximum value of the error, do not go to zero but approaches a constant value of 11 % of the size of the discontinuity. In the “Frequency Sampling Design” method, the overshoot may approach approximately 18% of the discontinuity [3].



**Figure 20 (a) Frequency response of an FIR filter affected by Gibbs phenomenon (b) Power Spectrum of an FIR filter affected by Gibbs phenomenon**

#### 4.2.3 “Design by Windowing” filter module

The “Design by Windowing” method begins with a desired frequency response that can be represented as

$$H_d(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h_d[n]e^{-j\omega n} \quad (77)$$

where  $h_d[n]$  is the corresponding impulse response sequence. Let  $H_d(e^{j\omega})$  be an ideal low-pass filter with frequency response

$$H_d(e^{j\omega n}) = \begin{cases} 1 & 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases} \quad (78)$$

where  $M$  is the filter order. As in the “Frequency Sampling Design” method, the impulse response of the ideal frequency response can be obtained by performing an inverse Fourier transform.

$$h_d[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(e^{j\omega n}) d\omega \quad (79)$$

However, to improve the impulse response of the filter and to reduce Gibbs phenomenon the ideal impulse response is truncated using a window. The simplest way to obtain a causal FIR filter from  $h_d[n]$  is to define a new system with impulse response  $h[n]$  given by

$$h[n] = \begin{cases} h_d[n] & 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases} \quad (80)$$

where  $M$  is the order of the transfer function polynomial. Thus,  $(M + 1)$  is the length of the impulse response. Alternatively, we can represent  $h[n]$  as the product of the desired impulse response and a finite-duration “window”,  $w[n]$ ,

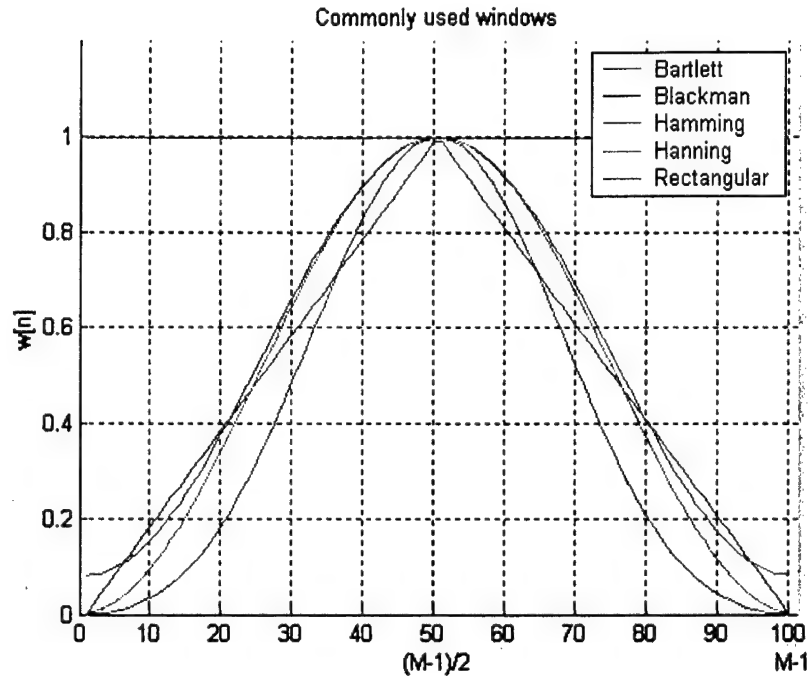
$$h[n] = h_d[n]w[n] \quad (81)$$

This multiplication truncates the ideal infinite impulse response,  $h_d[n]$ , to obtain a finite impulse response,  $h[n]$ , with less imperfection, thus reducing the effects of Gibbs phenomenon. In the frequency domain, Equation (81) can be expressed as

$$H(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(e^{j\omega}) W(e^{j\omega-\theta}) d\theta \quad (82)$$

That is,  $H(e^{j\omega})$  is the periodic convolution of the desired ideal frequency response with the Fourier transform of the window. Thus, the frequency response  $H(e^{j\omega})$  will be a “smeared” version of the desired response  $H_d(e^{j\omega})$ .

Some commonly used windows [1] are shown in Figure 21 and their equations listed in Table 6. The Hamming window used for this project is shown in red in Figure 21.

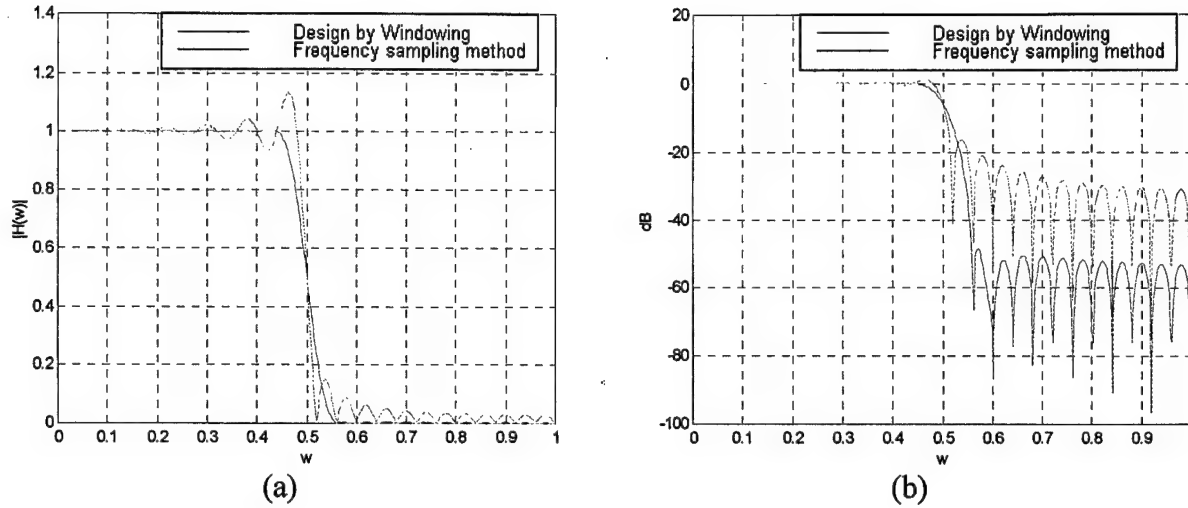


**Figure 21 Commonly used windows**

Window	Window equation
Rectangular	$w(n) = \begin{cases} 1, & 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases}$
Barlett (triangular)	$w(n) = \begin{cases} 2n/M, & 0 \leq n \leq M/2 \\ 2 - 2n/M & M/2 < n \leq M \\ 0 & \text{otherwise} \end{cases}$
Hanning	$w(n) = \begin{cases} 0.5 - 0.5 \cos(2\pi n/M) & 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases}$
Hamming	$w(n) = \begin{cases} 0.54 - 0.46 \cos(2\pi n/M) & 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases}$
Blackman	$w(n) = \begin{cases} 0.42 - 0.5 \cos(2\pi n/M) + 0.08 \cos(4\pi n/M) & 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases}$

**Table 6 Commonly used window functions**

Figure 22 shows the improvement obtained with the “Design by Windowing” method as compared with the “Frequency Sampling Design” method. The overshoot near the discontinuity, in the passband and in the stopband, has been considerably reduced using the “Design by Windowing” method.



**Figure 22** Comparison between the “Frequency Sampling Design” and “Design by Windowing” methods for FIR filter design. (a) Frequency response. (b) Power spectrum

The power spectrum on the right side shows that the attenuation in the stopband has been improved by more than 20 dB.

#### 4.2.4 Parks-McClellan filter module

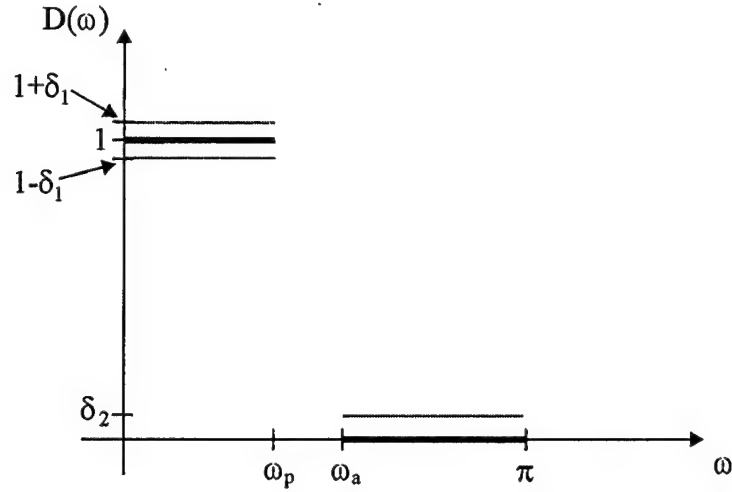
While the design of FIR filters with the “Frequency Sampling Design” method with or without windowing is straightforward, there are a number of limitations. The “Parks-McClellan” algorithm yields optimal filters and offers more control on certain regions of the frequency response of the filter. The Parks-McClellan algorithm is based on expressing the filter design problem as a problem in polynomial approximation [1] which is described briefly in the next section.

##### 4.2.4.1 Chebyshev approximation

Given the problem of designing a low-pass filter with specifications such as those shown in Figure 3, the Parks-McClellan algorithm considers that the desired frequency response of a filter may be approximated by a  $c$ 'th-order polynomial in  $\cos \omega$  as follows

$$P(\omega) = \sum_{k=0}^c a_k (\cos \omega)^k \quad (83)$$

where  $c = N/2$ ,  $N$  is the number of samples of the filter impulse response, and coefficients  $a_k$  are chosen so as to yield a  $P(\omega)$  which is optimal in a sense that is defined below.



**Figure 23 The desired frequency response  $D(\omega)$  of a low-pass filter**

Consider  $D(\omega)$  to be the desired frequency response, also shown in Figure 23, so that

$$D(\omega) = \begin{cases} 1 & \text{for } 0 \leq \omega \leq \omega_p \\ 0 & \text{for } \omega_a \leq \omega \leq \pi \end{cases} \quad (84)$$

and let  $W(\omega)$  be the weighting function for the approximation error over each of the intervals that the filter is defined,

$$W(\omega) = \begin{cases} 1 & \text{for } 0 \leq \omega \leq \omega_p \\ \delta_1 / \delta_2 & \text{for } \omega_a \leq \omega \leq \pi \end{cases} \quad (85)$$

where  $\delta_1$  and  $\delta_2$  are the amplitude of the passband and stopband ripple respectively. The error made by the approximation of  $D(\omega)$  by  $P(\omega)$  can be computed by [1][6][7]

$$E(\omega) = W(\omega)[D(\omega) - P(\omega)] \quad (86)$$

The weighted error function,  $E(\omega)$ , the weighting function  $W(\omega)$ , the desired frequency response  $D(\omega)$  and the approximation polynomial  $P(\omega)$ , are defined for the same discrete subset of frequencies taken from the interval  $0 \leq \omega \leq \pi$ . For a low-pass filter, these four functions will be defined over  $0 \leq \omega \leq \omega_p$  and  $\omega_a \leq \omega \leq \pi$  as indicated in Equations (84)

and (85). The subinterval  $[\omega_p, \omega_a]$  is the transition band of the filter. As  $E(\omega)$ ,  $W(\omega)$ ,  $D(\omega)$  and  $P(\omega)$ , are not defined over the transition band, the Parks-McClellan method [1] allows  $P(\omega)$  to take any shape in this band to achieve its optimum approximation and meet the filter specifications. The criterion used in this design procedure is the Chebyshev Criterion [1] where within the frequency intervals of interest,  $P(\omega)$  is chosen to minimize the maximum weighted approximation error. This can be expressed by

$$\min_{h[n]: 0 \leq n \leq L} \left\{ \max_{\omega \in F} |E(\omega)| \right\} \quad (87)$$

where  $F$  is the closed subset of  $0 \leq \omega \leq \pi$  over which the filter is specified (i.e.  $0 \leq \omega \leq \omega_p$  and  $\omega_a \leq \omega \leq \pi$ ). Thus, the method is to seek the set of frequencies that produce the impulse response values that minimize the error. These impulse response values are then used to compute the coefficients of a desired filter. An IDFT will be performed on the impulse response values to give the coefficients of the filter in the time-domain. A weighting factor may be associated with each frequency subinterval (i.e.  $0 \leq \omega \leq \omega_p$  and  $\omega_a \leq \omega \leq \pi$  for a low-pass filter). The general idea for using a weighting factor is to exaggerate the error of the approximation. The algorithm, in turn, will try to produce a better approximation for bands with the higher weighting factor by minimizing this amplified error. This results in more iterations to compute the approximation but a more accurate response is achieved in the higher weighted band.

Parks-McClellan applies the Alternation Theorem from approximation theory to minimize the error. The Alternation Theorem is described in the next section.

#### 4.2.4.2 Alternation theorem

The Alternation Theorem states that if the frequency response of a filter is represented by a linear combination of  $c$  cosine functions, as expressed in Equation (83), the best-weighted Chebyshev approximation to a desired frequency response,  $D(\omega)$  is achieved if the weighted error function  $E(\omega)$  exhibits at least  $c+2$  extremal frequencies in  $F_p$ . An extremal frequency is the frequency at which a ripple (either in the passband or the stopband) is at its maximum or minimum value.  $F_p$  is the subset of frequency intervals over which the passband and stopband are defined. Thus, there are at least  $c+2$  frequencies in  $F_p$  such that  $\omega_1 < \omega_2 < \dots < \omega_{c+2}$  and such that  $E(\omega_i) = -E(\omega_{i+1}) = \pm E$  for  $i = 1, 2, \dots, c+1$  [1]. To find the extremal frequencies from the discrete subset of potential extremal frequencies  $F_p$ , the Parks-McClellan method uses the Remez Exchange algorithm. The Remez Exchange algorithm is a set of conditional statements applied to the potential extremal frequencies contained in  $F_p$ . Based on the Chebyshev error criterion, the Remez Exchange algorithm will determine if a frequency is an extremal frequency or not. The  $c+2$  extremal frequencies

found by the algorithm are then used to compute the filter frequency response with the approximation function  $P(\omega)$ . The Remez Exchange algorithm is described further in [6][7].

From the Alternation Theorem we can write [7]

$$E(\omega_i) = W(\omega_i)[D(\omega_i) - P(\omega_i)] = (-1)^{i+1} \delta, \quad i = 1, 2, \dots, (c+2) \quad (88)$$

where  $\delta$  is the optimum error. Parks and McClellan [1][6][7] found that for the given set of the extremal frequencies,  $\delta$  is given by the formula

$$\delta = \frac{\sum_{k=1}^{c+2} b_k D(\omega_k)}{\sum_{k=1}^{L+2} \frac{b_k (-1)^{k+1}}{W(\omega_k)}} \quad (89)$$

where

$$b_k = \prod_{\substack{i=1 \\ i \neq k}}^{c+2} \frac{1}{(x_k - x_i)} \quad (90)$$

and

$$x_i = \cos \omega_i \quad (91)$$

Parks and McClellan used the Lagrange interpolation formula [1][6] to obtain

$$P(\omega) = \frac{\sum_{k=1}^{c+1} [d_k / (x - x_k)] C_k}{\sum_{k=1}^{L+1} [d_k / (x - x_k)]} \quad (92)$$

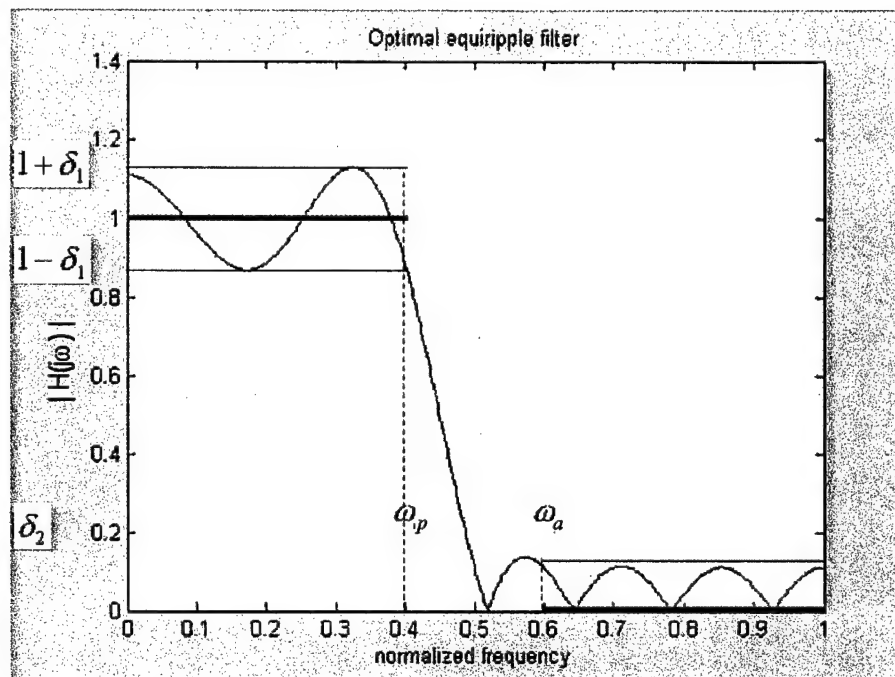
where

$$x = \cos \omega \quad (93)$$

$$C_k = D(\omega) - \frac{(-1)^{k+1} \delta}{W(\omega)} \quad (94)$$

$$d_k = \prod_{\substack{i=1 \\ i \neq k}}^{c+1} \frac{1}{(x_k - x_i)} = b_k (x_k - x_{L+2}) \quad (95)$$

Figure 24 shows an example of a low-pass filter described by the Parks-McClellan method. A flow diagram of the algorithm can be found in [1].



**Figure 24** Typical example of a low-pass filter approximation that is optimal according to the alternation theorem for  $c = 7$

#### 4.2.5 Gaussian digital filter module

A digital GMSK modulator is to be implemented in the ROBR testbed. One implementation of the GMSK modulator requires a Gaussian filter and a digital integrator as shown in Figure 25. As a result, a Gaussian filter is also implemented using an FIR filter design technique. The coefficients computation module uses the “Frequency Sampling Design” method to calculate the coefficients of a Gaussian filter. However, instead of sampling the ideal frequency response of a low-pass filter, a Gaussian distribution is sampled. The impulse response of a Gaussian filter is expressed as [12]

$$h(t) = \frac{k_1 B}{\sqrt{\pi}} e^{-k_1^2 B^2 t^2}, \quad \text{where } k_1 = \frac{\pi}{\sqrt{2 \ln 2}} \quad (96)$$

or

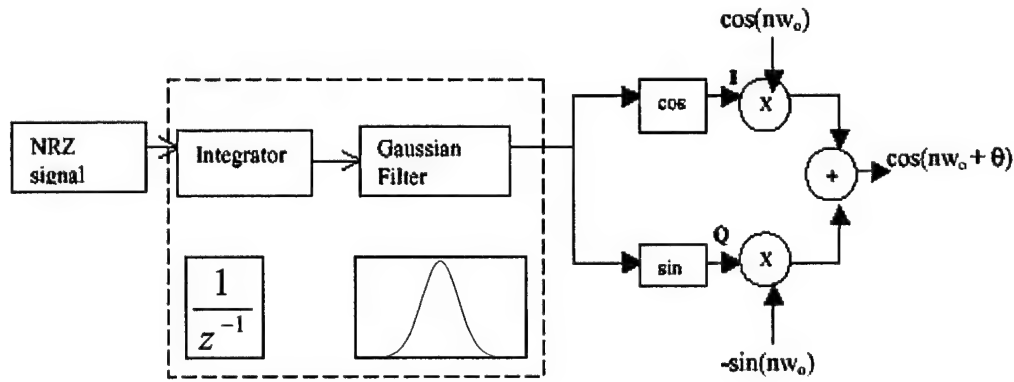


$$h(t) = \frac{1}{\sqrt{2\pi}\sigma T} e^{\left[\frac{-t^2}{2\sigma^2 T^2}\right]}, \quad \sigma = \frac{\sqrt{\ln 2}}{2\pi BT} \quad (97)$$

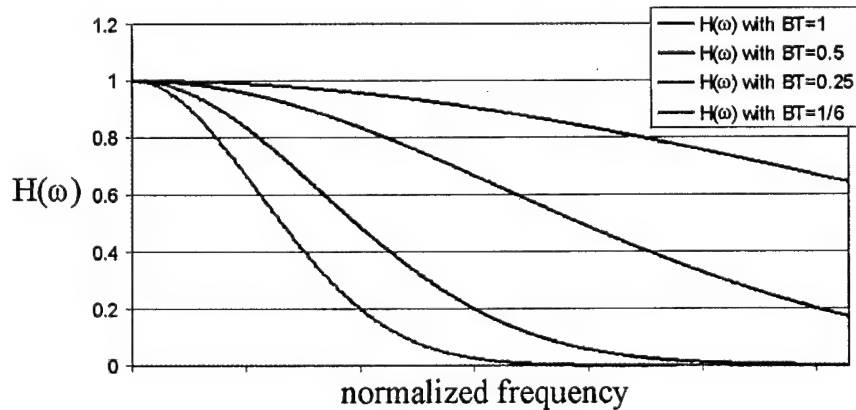
where  $B$  is the 3dB bandwidth of the Gaussian filter and  $T$  is the bit period. The sampling of the Gaussian distribution is done over the frequency domain expression obtained by taking the Fourier Transform of Equation (96) giving

$$H(\omega) = e^{-\frac{1}{4} \frac{\omega^2}{k^2 B^2}} \quad (98)$$

Figure 26 shows Gaussian frequency response for various values of  $BT$  product.



**Figure 25** The premodulator stage of GMSK digital modulator includes a digital integrator followed by a Gaussian filter



**Figure 26** Frequency response for a Gaussian filter

#### 4.2.5.1 Digital integrator module

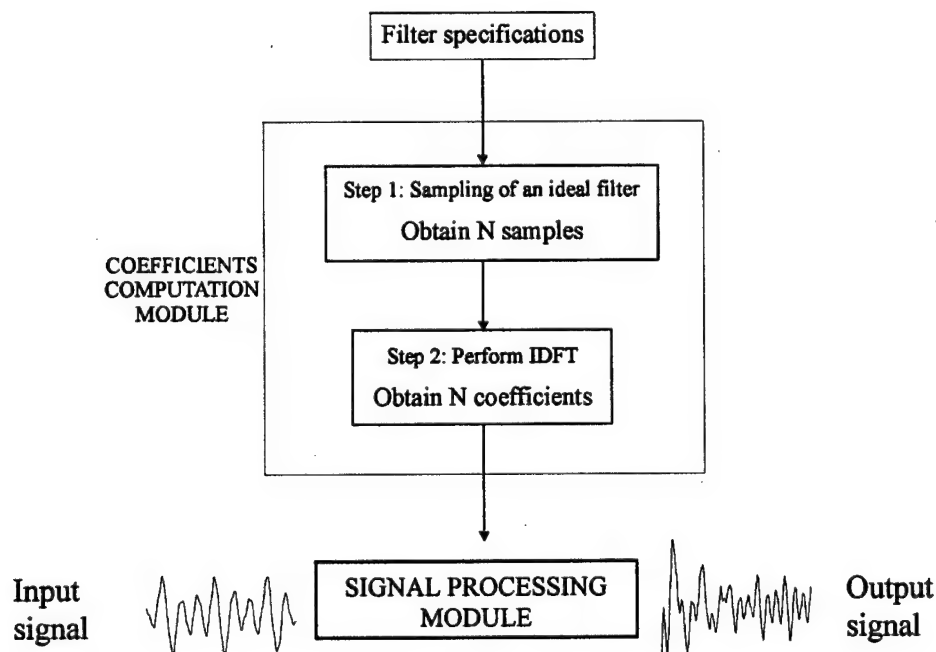
As mentioned in Section 4.2.5, a GMSK modulator is to be implemented in the ROBR and requires a digital integrator. As a result, the implementation of a digital integrator module is included here. The implementation of the digital integrator is very simple. The output for a given moment,  $y_k$ , of such a filter is the summation of the present input  $x_k$  and all the previous or past inputs, which can be expressed mathematically as

$$y_k = \sum_{i=0}^k x_i \quad (99)$$

### 4.3 FIR filter module implementation

#### 4.3.1 System

As with the IIR filter modules, the FIR filter modules implemented consist of two components, a coefficients computation module and a signal processing module. The coefficients computation module is subsequently broken down into two stages, a sampling step and the computation of an inverse discrete Fourier Transform. The coefficients computation module takes the filter specifications, computes the desired FIR design and generates the coefficients of the filter. The signal processing module takes an input signal and processes it using the coefficients computed by the coefficients computation module. Figure 27 shows the architecture of the FIR filter modules.



**Figure 27 Block diagram of FIR filter module implementation**

The three design methods described earlier have been implemented to compute FIR filter coefficients. The “Frequency Sampling Design” method, the “Design by Windowing” method and the “Parks-McClellan” method are described in the following sections. In addition, the implementation of filter modules for the Gaussian filter and the digital integrator are presented.

#### 4.3.2 FIR coefficients computation modules

For FIR filters, the coefficients computation modules generate the coefficients  $h_k$  identified in Equation (73). The coefficients are subsequently used in the FIR signal processing modules to process a digital signal. Five FIR coefficient computation modules have been implemented. The implemented modules are listed in Table 7.

Filter Design	Module Name
Frequency Sampling Design method	Freqsampling
Design by Windowing method	Hamming
Park-McClellan Method	Remezex
Gaussian filter	Gauss
Digital Integrator	Dintegrator

**Table 7 FIR filter types implement by the corresponding modules**

More information about how to use each module can be found in the user’s guide included with the software developed. All FIR coefficient computation modules, except the digital integrator module, require two input parameters: the digital cutoff frequency and the sampling frequency.

##### 4.3.2.1 Frequency sampling design method module implementation

The design method starts in the frequency domain and is separated into two steps. To apply the “Frequency Sampling Design” method, a vector is created which contains the frequency response samples of the desired low-pass filter. The size of this vector will be the same as the number of coefficients to be computed. This operation is equivalent to sampling an ideal frequency response. Once this vector is built, an IDFT is then performed on the vector’s elements to compute the coefficients of the filter in the time domain. The implementation of these two steps to design a low-pass filter is described in the next sections.

##### 4.3.2.1.1 Sampling the ideal frequency response

The first step in designing an FIR filter is to build the vector of frequency response samples. For a low-pass filter, the elements which belong to the passband of the ideal frequency response will be set to ‘1’ and the elements which belong to the stopband of the ideal

frequency response will be set to '0'. The number of elements set to '1' or '0' will depend on the normalized cutoff frequency and on whether the number of coefficients is odd or even. The number of elements to set to '1' in the frequency response vector of a low-pass filter can be computed [5] using the program code shown in Figure 28.

```
if(num_taps%2)                //odd number of coefficients case
    numsamples = ceil(num_taps * Wd/(2*PI) - 0.293);
else
    numsamples = ceil(num_taps * Wd/(2*PI) - 0.207);
```

**Figure 28 Computation of the number of samples to include in the passband of an ideal low-pass filter**

The other elements of the vector will be set to '0' to represent the stopband. The creation and initialization of this vector is equivalent to sampling an ideal frequency response because each element of this vector may be viewed as a sample. The program code needed to build this vector is shown in Figure 29. Thus, an example of a frequency response vector could be  $H[n] = [1,1,1,1,1,0,0,0,0,0,]$  for a low-pass filter with 10 coefficients.

```
//sampling of the ideal frequency response

for(int i=0;i<numsamples;i++)
{
    H[i] = 1.0;
}

for(int i=numsamples;i<num_taps;i++)
{
    H[i] = 0.0;
}
```

**Figure 29 Sampling of an ideal low-pass filter frequency response**

#### 4.3.2.1.2 Implementation of the IDFT

The IDFT is performed on the frequency domain samples of the ideal low-pass filter generated in the previous section. The appropriate expression of the IDFT is used depending on whether the number of coefficients is odd or even. The algorithm is listed in Figure 30.

```

Float temp = 0;
Float mid_pt = (num_taps-1.0)/2.0;
Float x;
if(num_taps%2)           // N odd
{
    for(int n=0;n<num_taps;n++)
    {
        temp = H[0];
        x = 2 * PI * (n - mid_pt)/num_taps;
        for(int k=1;k<((num_taps-1)/2);k++)
            temp+=(2.0*cos(x*k))*(H[k]);
        h[n] = temp/num_taps;
        fprintf(outputfile,"%f\n",h[n]); //printing the
                                         coefficients in an output file
    }
}
else           // N even
{
    for(int n=0;n<num_taps;n++)
    {
        temp = H[0];
        x = 2 * PI * (n - mid_pt)/num_taps;
        for(int k=1;k<=(num_taps/2) - 1;k++)
            temp+=(2.0*cos(x*k))*(H[k]);
        h[n] = temp/num_taps;
        fprintf(outputfile,"%f\n",h[n]);
    }
}

```

**Figure 30 Inverse discrete Fourier transform algorithm performed on the ideal low-pass filter frequency response samples**

#### 4.3.2.1.3 Design by windowing

The only difference between the “Design by Windowing” method and the “Frequency Sampling Design” method is that, for the design by windowing method, the result of the IDFT will be multiplied by a vector containing the samples of a Hamming window. The equation to compute the samples of a Hamming window is given in Table 6 in Section 4.2.3. The computed samples of the Hamming window will be stored in a vector of N elements, where N is the number of filter coefficients. Then the coefficients obtained from the computation of the IDFT, and stored in  $h[n]$ , will be multiplied by this vector, as shown in Figure 31.

```

.
.
.
for(int k=1;k<((num_taps-1)/2);k++)
    temp+=(2.0*cos(x*k))*(H[k]);

//multiplying the time domain coefficients by the Hamming window
h[n] = temp/num_taps * (0.54 - 0.46*cos(2*PI*n/num_taps));

//printing the coefficients in an output file
fprintf(outputfile,"%f\n",h[n]);
.
.
.

```

Figure 31 “Design by Windowing” algorithm

#### 4.3.2.2 Parks-McClellan method implementation

The implementation of the Parks-McClellan FIR coefficients computation module is included in the package *digital\_filters* under the directory named *remezex*. The implementation is separated into two files: *remezex.c* and *remez.c*. The library *remez.h* is required by these two files and contains the appropriate constants and functions used by them. The file *remez.c* and *remez.h* are external files that were first created by Jake Janovetz and can be used under the GNU General Public license restrictions. The source code implemented by Jake Janovetz provides a very good implementation of the Remez Exchange algorithm and forms the basis of the implementation of the reconfigurable module. The GNU General Public License allows anyone to use and modify a file or a portion of code placed under the terms of this license. The GNU General Public License has been included in the *digital\_filters* package. To find out more about the terms and agreements of this license, the reader is referred to <http://www.gnu.org/licenses/gpl.html> on GNU’s web site.

The task of the file *remezex.c* is to acquire the filter design parameters entered by the user, initialize the variables used by the module and make the function calls to compute the filter coefficients. It is noted that *remezex.c* will normalize the input digital frequency over the interval [0,0.5], as is required for the implementation of the Parks-McClellan method [3]. Then, the file *remezex.c* fills three arrays depending on the design parameters entered.

The array *desired[]* contains the magnitude of the sampled ideal frequency response. For a low-pass filter, there are two elements in the array *desired[]* to represent the magnitude of the frequency response in the passband and in the stopband respectively. In this implementation, the first element is initialized to ‘1’ to represent the magnitude of the frequency response in the passband. The second element is initialized to ‘0’ to represent the magnitude of the response in the stopband.

The array *weights[]* contains floating point values representing the weights given to the passband(s) and stopband(s). These weights will determine the importance to give to the corresponding frequency band in the computation of the algorithm. A large value given to a weight corresponding to a specific band will put the emphasis of the algorithm on producing a better approximation of the ideal frequency response in this band compared to the approximation made in the other bands [1]. For a low-pass filter, the *weights[]* array contains two elements. The first one specifies the weighting assigned to the passband and second specifies the weighting assigned to the stopband. For this implementation, each frequency band was given the same weighting and thus, both elements are set to '1'.

The array *bands[]* contains the edge frequencies which delimit the passband and the stopband of the low pass filter's frequency response. This array contains four normalized frequency values to delimit the passband and the stopband and should be in the form ,  $[0, \omega_p, \omega_s, 0.5]$ . In this implementation, the user enters the desired normalized cutoff frequency,  $W_d$ , and a value of  $\omega_p = W_d - 0.025$  and  $\omega_s = W_d + 0.025$  is computed for the array *bands[]*. The length of the transition band between the passband and the stopband, for this module, has been arbitrarily chosen to be 0.05 Hz/Hz.

Figure 33 shows the contents of *remezex.c*. After, filling the three required arrays, *remezex.c* makes a function call to *void remez()*, the function which computes the filter coefficients using the Remez Exchange algorithm. *void remez()* is implemented in the file *remez.c*.

```
desired[0] = 1;
desired[1] = 0;

weights[0] = 1;
weights[1] = 1;

bands[0] = 0;
bands[1] = Wd-0.025;
bands[2] = (Wd + 0.025);
bands[3] = 0.5;

remez(&h[0], num_taps, 2, bands, desired, weights, BANDPASS);
```

**Figure 32** Content of the file *remezex.c*

#### 4.3.2.3 Gaussian filter implementation

The implementation of the Gaussian filter coefficients computation module is based on the "Frequency Sampling Design" method. Instead of computing the samples of the frequency response of an ideal low-pass filter, samples for a Gaussian distribution are calculated based

on Equation (98). Figure 32 shows the portion of code that is used to fill in the array of frequency response samples,  $H[i]$ .

```
//computing the sampling step
step = (float) (Fs/num_taps);

//sampling of the gaussian curve
for(int i=0;i<num_taps;i++)
{
    H[i] = exp((-1.0/4.0)*(2.0*PI*i*step)*
               (2.0*PI*i*step)/(K*K*B*B));
}
```

**Figure 33 Sampling algorithm of a Gaussian distribution**

The next step of the implementation is to compute the IDFT of the sampled curve to get the filter coefficients as was done for the “Frequency Sampling Design” method implementation and the “Design by Windowing” method implementation.

### 4.3.3 FIR signal processing modules

The FIR signal processing module has been implemented based on the convolution equation introduced in Section 4.1. This implementation is based on the Direct Form structure presented in Section 4.1.1. The signal processing modules can be found in the package of filter modules under the name of *firdf*.

Figure 34 shows the implementation of the algorithm of the FIR signal processing module. A buffer is used to store the input samples required for the convolution. First, the previous input samples are shifted down in the buffer. The current input sample is then read from an input text file and stored at the beginning of the buffer. Then, a convolution sum, as described in Equation (4), is performed to produce an output sample which is stored in an array. The output array is printed to a text file once all the input samples have been processed. This implementation is able to operate in real time as each input sample is read and processed to directly produce an output sample.



```

while(fscanf(inputfile,"%f",&sample)!=EOF)
//reading input sample from the input file
{
    for(j=numcoef-1;j>0;j--)
    {
        if(indx-j < 0) continue;

        inputBuffer[j]=inputBuffer[j-1];
    }
    inputBuffer[0]=sample;

//convolution operation to produce output sample

    output=0;
    for(i = 0; i< numcoef;i++)
    {
        if(indx-i < 0) break;
        output+=coeff[i]*inputBuffer[i];
    }
    fprintf(outputfile,"%f\n",output);
    indx++;
}

```

**Figure 34 FIR signal processing module algorithm**

#### **4.3.4 Use of the filter and signal processing modules**

The following is an example of how the modules are used to generate coefficients for a low-pass Butterworth filter. The coefficients computation modules can be executed from the prompt of a console as shown in Figure 35. The module *butter* takes four arguments as input design specifications; the filter's order, the normalized cutoff frequency, the sampling frequency and the number associated with the method used to compute the discrete design. The filter order is entered as an integer value. The normalized cutoff frequency is entered as a floating point number between 0.0 and 1.0 for the Butterworth filter. The sampling frequency is a floating point number and is normalized to 2.0 in this example. The units of the normalized cutoff frequency and the sampling frequency are Hz/Hz. It is noted that the normalized cutoff and sampling frequencies are relative. As a result, the filter module allows for scalability of the filter design and generates the same coefficients for filters with the same ratio of cutoff frequency to sampling frequency. The last parameter specifies the method used to compute the discrete design. A '1' is entered to select the Impulse Invariance method, and a '2' is entered to select the Bilinear transformation.

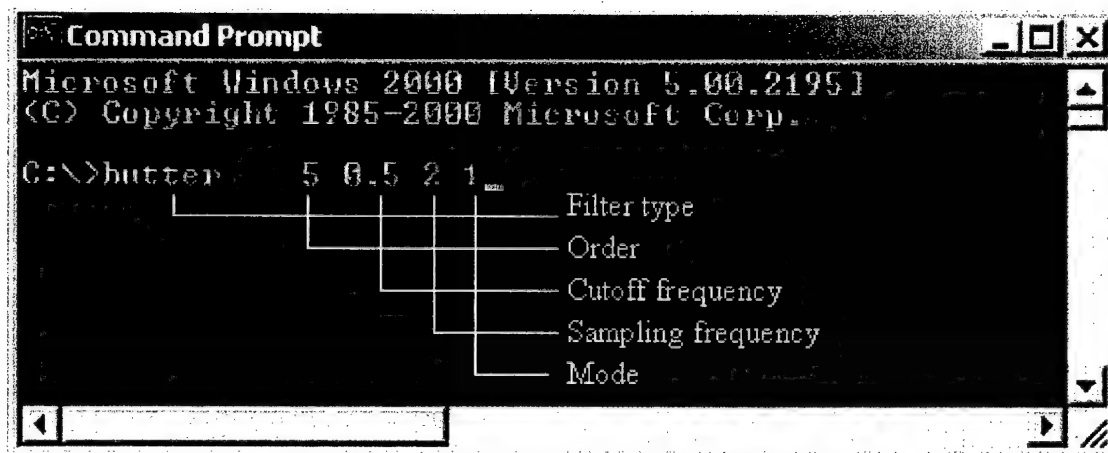


Figure 35 Command line for Butterworth filter module

After the computation of the coefficients is complete, the module generates an output file, "butterout.txt" with the coefficients of the filter. Figure 36 shows an example of the coefficients computation module output file. The order of the filter computed is printed on the first line. Then, on the second line, the digital static gain is printed and finally, the coefficients of the filter are printed in four columns. The two left most columns contain the real and imaginary parts of the numerator coefficients ( $b_k$ ). The two right most columns contain the real and imaginary parts of denominator coefficients ( $a_k$ ).

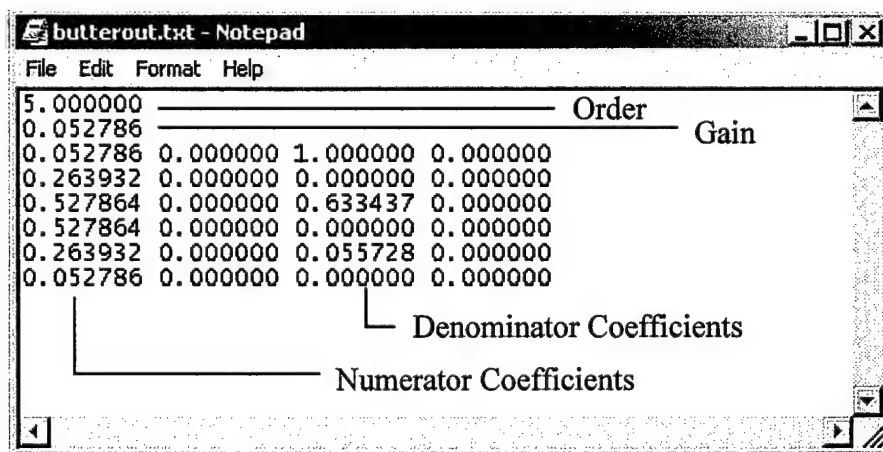
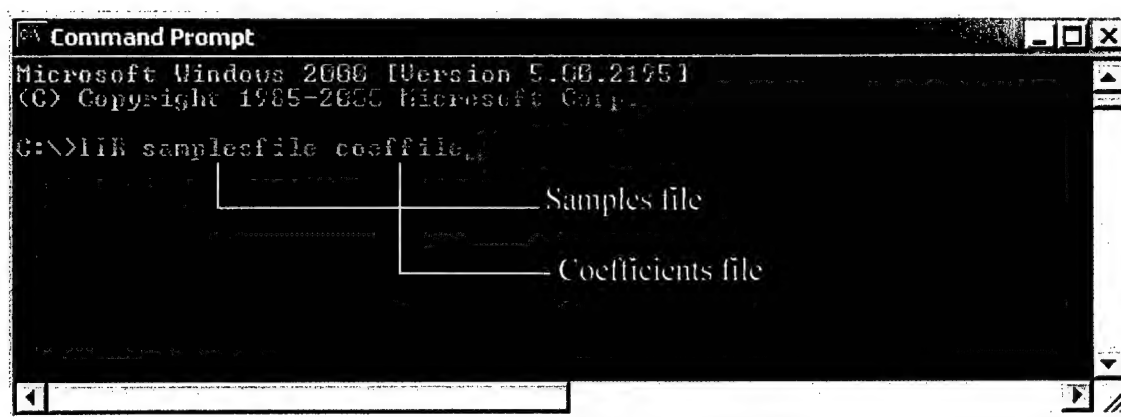


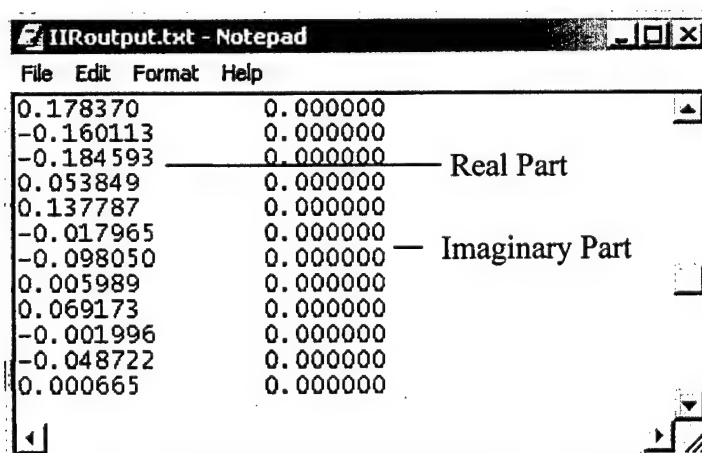
Figure 36 Example of an output file generated by the coefficients computation modules.

The signal processing modules can be executed from a console prompt as in the case of the coefficients computation modules. Figure 37 shows an example of how to execute the signal processing module for an IIR filter. The first argument of these modules is the file name of the input samples. The second argument is the name of the text file which contains the coefficients of the implemented filter design. The coefficients file is the output file obtained from the coefficients computation modules (i.e. "butterout"). The user's guide contains information on the format of the input data file.



**Figure 37** Command line for executing an IIR signal processing module.

The output file generated by the signal processing module contains the output of the filter used to process the input samples file. One output sample is printed on each line of the file as shown in Figure 38. The first column contains the real part of the output and the second column contains the imaginary part of the output. In the example shown in Figure 38, the values of the imaginary part of the results are zero as a real signal was processed rather than a complex signal.



**Figure 38** Example of an output file generated by a signal processing module

## 5 DSP IMPLEMENTATION

The implementations of the filter modules described in the previous sections have been targeted for a general purpose processor. In order to make use of these digital filter modules in the ROBR and take advantage of their reconfigurability in “real-time”, a DSP implementation of the modules is required. For the DSP implementation, a DSP board manufactured by Spectrum Signal Processing, Inc., was used. The DAYTONA DSP board is a PCI dual processor board and contains two TMS320C6201 fixed point DSP chips. Only one processor is used for the filter module implementation. Code development for the DSP board is done in the ANSI C programming language. A compiler, code generator, and linker are provided with the DSP board.

Two FIR coefficient computation modules and one FIR signal processing module have been adapted for the DAYTONA DSP board. In all three cases, since the original module was written in C, only minor changes were required to yield compatibility between the filter modules and the DSP architecture. The coefficients computation modules implemented are the “Design by Windowing” method and the “Parks-McClellan” method. The FIR signal processing module implements the Direct Form structure shown previously in Section 4.1.1.

The DSP board is hosted in a PC with the WIN NT operating system. The host handles the initialization, handshaking and downloading of processor code through the PCI bus. As such, the integration of the filter and signal processing modules requires two different files: a host program that controls the DSP and provides the user interface, and the DSP file which implements the DSP program is responsible for processing the data.

### 5.1 Exchanging data between the Daytona and the host station

The DSP processor has both internal and external memory spaces available including internal program and data RAM, external SSRAM, external SDRAM, and dual port RAM. For this project, the memory used for the data exchange is the SDRAM of processor 0 on the Daytona DSP board [9]. The SDRAM block of the processor goes from address 0x0200000 to address 0x02FFFFFF. The other memory spaces available on the DSP board are listed in Table 8.

Description	Size	Internal or External
Program RAM	64kB	Internal
SSRAM	16MB	External
I/O, boot	4MB	External
SDRAM	16MB	External
Processor Expansion Module (PEM)	64kB	External
Internal Registers	256kB	Internal

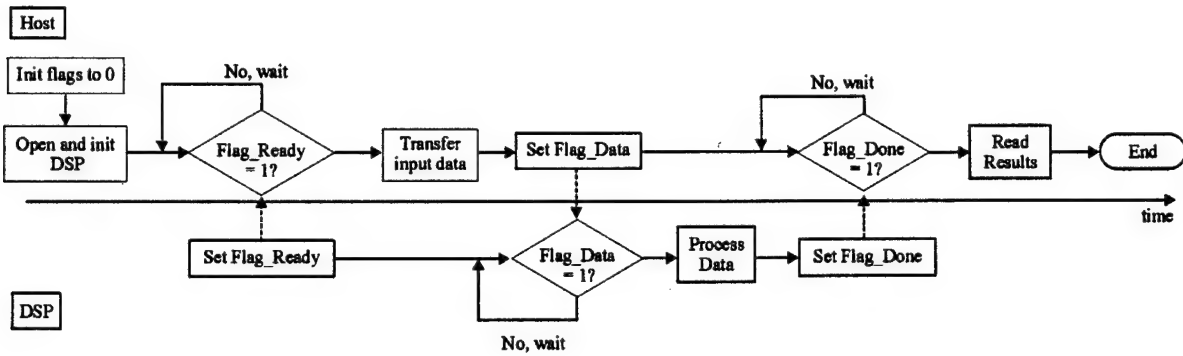
**Table 8 Memory configuration of the TMS320C6201**

## 5.2 Static and determined length memory allocation

The static allocation of memory is done on the SDRAM block of the processor. Figure 39 shows the memory allocation for a typical filter module. Four 32-bit variables are used as flags for each module. They are *Flag\_Ready*, *Flag\_Data*, *Flag\_Done*, and *OK\_Memory*. The addresses for the memory space are assigned in bytes so that consecutive 32-bit words are addressed 4 bytes apart. The use of flags makes the synchronization between the host program and the DSP possible. The flag *Flag\_Ready* is set by the DSP to tell the host program that the DSP is ready to compute data. The flag *Flag\_Data* is set by the host to signal to the DSP that data is ready to be transferred from the host. The flag *Flag\_Done* is set by the DSP to tell the host program that the DSP has finished its computations. The flag *OK\_Memory* is used by the DSP to tell the host program that the memory needed for the computation has been successfully allocated. As shown in Figure 39 memory has been allocated for *Num\_Input* and *Num\_Coeff* which are integer values that are used to store the number of coefficients to be computed and the number of input data samples. Arrays are allocated at addresses pointed to by *Coeff* and *Data*. The array *Coeff* contains the coefficients computed by the DSP that are transferred back to the host program. The array *Data* is used to hold the input signal samples to be transferred from the host to the DSP. After the DSP has performed the signal processing computation, the same area is used to store the computed output signal. The host program will then be able to retrieve the output signal from the array *Data*. Figure 40 shows a flow diagram of the handshaking between the host and the DSP.

```
#define Flag_Ready (UINT32*)(0x02000004)
#define Flag_Data (UINT32*)(0x02000008)
#define Flag_Done (UINT32*)(0x0200000c)
#define Num_Input (UINT32*)(0x02000010)
#define Num_Coeff (UINT32*)(0x02000014)
#define OK_Memory (UINT32*)(0x02000018)
#define Coeff (float*)(0x0200001c)
#define Data (float*)(0x02000090c)
```

**Figure 39 Static allocation of the variables in the SDRAM memory**



**Figure 40 Flow diagram of the implementation of the filter modules**

### 5.3 Dynamic memory allocation

The function call *malloc()* is a service provided by the run-time support library included in the DSP compiler. *malloc()* extensively used in the reconfigurable modules because the size of the arrays needed to store the data from the user is not known in advance. Memory is dynamically allocated from a memory space defined in the “sysmem” memory section of the DSP. The sysmem memory section is created and allocated prior to the compilation and the linking of the program in the link command file. In Figure 41, a portion of the memory map generated by the linker shows that the “sysmem” memory section, which is 0x3000 bytes (or 12kB) in size, is allocated to begin at address 0x80002018.

```

*****
TMS320C6x COFF Linker Version 2.00
*****

```

# MEMORY CONFIGURATION

name	origin	length	used	attributes	fill
IVECS	00000000	000000400	00000200	RWIX	
Iprog	00000400	00000fc00	00001b80	RWIX	
SSRAM	00400000	000400000	00000000	RWIX	
MPRAM	01400000	000200000	00000000	RWIX	
DL3	01600000	0000c0000	00000000	RWIX	
IREG	01800000	000800000	00000000	RWIX	
SDRAM	02000000	001000000	00000000	RWIX	
PEM	03000000	001000000	00000000	RWIX	
IVARS	80000000	0000c0000	000050e8	RWIX	
IDATA	8000c0000	000004000	00000000	RWIX	

## SECTION ALLOCATION MAP

output section	page	origin	length	attributes/ input sections
.vectors	0	00000000 00000000	00000200 00000200	isfp6201.o6x (.vectors)
.text	0	00000400 00000400	00001b80 000009c0	rts6201.lib : memory.obj (.text)
		.	.	
.stack	0	80000000 80000000	00002000 00000000	UNINITIALIZED rts6201.lib : boot.obj (.stack)
.tables	0	80000000	00000000	UNINITIALIZED
.data	0	80000000 80000000	00000000 00000000	UNINITIALIZED FIRmod.o6x (.data)
		.	.	
.bss	0	80002000 80002000	00000014 00000008	UNINITIALIZED rts6201.lib : exit.obj (.bss:c)
		.	.	
.sysmem	0	80002018 80002018	00003000 00000000	UNINITIALIZED rts6201.lib : sysmem.obj (.sysmem)
.cinit	0	80005018 80005018	00000054 0000001c	rts6201.lib : sysmem.obj (.cinit)
.const	0	80000000	00000000	UNINITIALIZED

## GLOBAL SYMBOLS

Address	name	address	name
-----	----	-----	----

Figure 41 Sysmem memory section allocation in the DSP memory map.

## 5.4 Host program

The major function of the host program is to control the DSP board and act as the user interface to the DSP board. The host program is responsible for reading the design specifications of the filter module and passing them to the DSP. The Daytona Windows NT Host Application Library (ALIB\_HOST) provides several high-level functions that allow the user to control the operations of the Daytona from a Windows NT host.

### 5.4.1 Host software functions

The library ALIB\_HOST implements several host functions to control the Daytona board. The following functions have been used in the filter modules' implementation.

FT_Control()	: to reset the board
FT_ErrorMessage()	: to catch error message
FT_GetHandle()	: to get handle to Daytona system resources
FT_Read()	: to read from a system resource or host buffer
FT_SystemClose()	: to close the DSP board
FT_SystemLoad()	: to load the DSP code into the system
FT_SystemOpen()	: to open the system
FT_Write()	: to write to the system resources

## 5.5 DSP program

The DSP program contains all the data processing instructions. The algorithm of the original modules has not been modified but the variables declared were changed into pointers in most cases to be integrated for use on the DSP.

### 5.5.1 DSP software functions

As in the host program, library functions are provided for initialization, interrupts, and DMA transfers. In this project, the only library function called in the DSP code is C6x\_OpenC6x(). This function initializes the C6x processor, sets the wait states for external memory and configures the page register. The page register contains the addresses of the memory spaces that are available for each processor. The memory spaces are accessed through the PCI bus [11].



## 6 RESULTS AND VERIFICATION

### 6.1 Methodology

The implementation of the filter modules has been verified with the *Signal Processing* toolbox from MATLAB. Table 9 and Table 10 show which corresponding functions from MATLAB have been used to verify the coefficients generated by the reconfigurable filter modules.

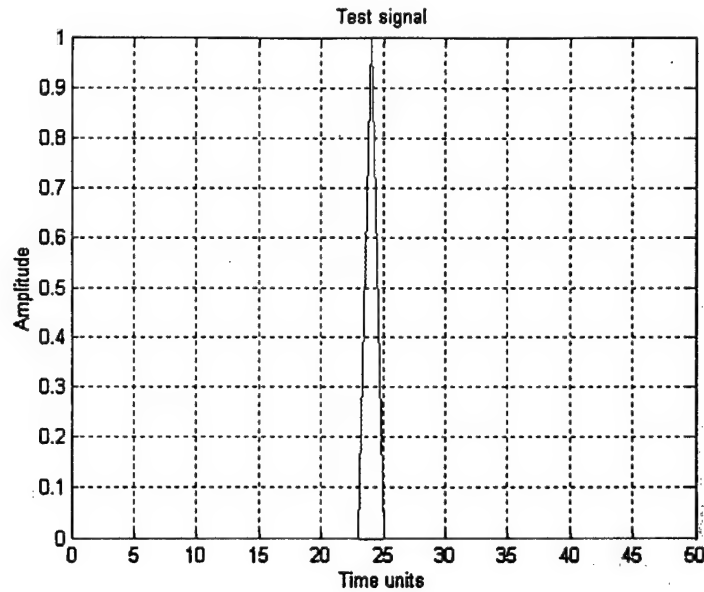
IIR coefficient computation modules		
Filter types	Reconfigurable Filter modules	Function from MATLAB's toolbox <i>signal</i>
Butterworth	butter	Butter
Chebyshev	cheb1	cheby1, cheblap
Inverse Chebyshev	cheb2	cheby2, che2ap
Elliptical	ellip	ellip, ellipap
Bessel	bessel	-

**Table 9 Corresponding MATLAB functions for the IIR modules verification**

FIR coefficient computation modules		
Design method used and Filter types	Reconfigurable Filter modules	Function from MATLAB's toolbox <i>signal</i>
Frequency sampling	freqsampling	-
Windowing	hamming	fir1 with hamming window
Parks-McClellan (optimal equiripple filter)	remezex	Remez
Gaussian filter	gauss	-
Digital Integrator	dintegrator	-

**Table 10 Corresponding MATLAB functions for the IIR modules verification**

The signal processing modules for the IIR and FIR filters were compared with the corresponding *filter* function from the *Signal Processing* toolbox. Coefficients generated by the coefficient computation modules were provided to the appropriate signal processing module to compute the response to an impulse. Subsequently, an FFT is performed using MATLAB's *FFT* function on the output of the signal processing module which gives the frequency response. Samples of the impulse function are used as the input to the signal processing module. The samples are generated using MATLAB and are composed of 49 "zeroes" followed by a "one", followed by 49 "zeroes" in floating point representation. Figure 42 is a plot of the impulse signal generated with MATLAB.



**Figure 42 Impulse signal generated with Matlab**

Thus, the verification process was carried out as follows:

- 1- Computation of the coefficients with the coefficients computation modules
- 2- Computation of the coefficients with MATLAB's toolbox
- 3- Comparison of the coefficients generated by the reconfigurable modules with the coefficients generated by MATLAB
- 4- Processing of an impulse with the signal processing modules using the coefficients computed by the coefficients computation modules
- 5- Processing of an impulse with the MATLAB's function using the coefficients computed by MATLAB
- 6- Performing an FFT on both output generated by the filter modules and by MATLAB functions.
- 7- Comparison of the frequency response yielded by the filter modules and by MATLAB.

## **6.2 Results**

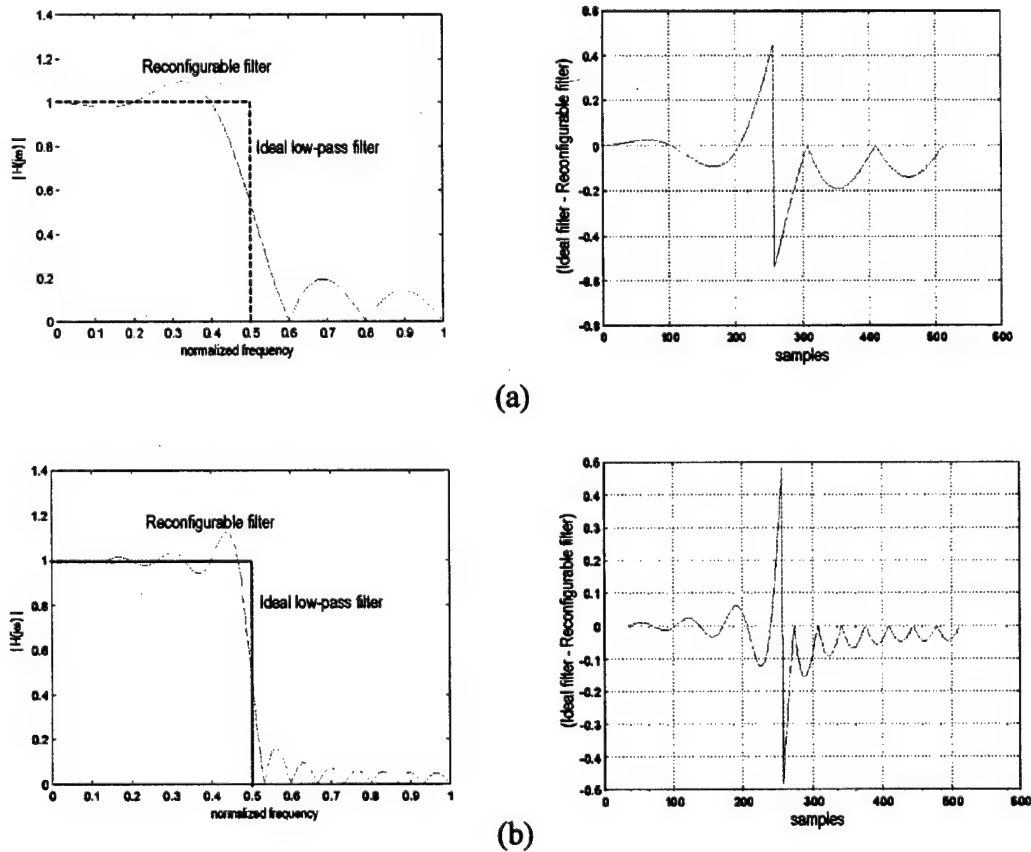
### **6.2.1 IIR filter module verification**

Appendix A shows the plots comparing the frequency response of the Butterworth, Chebyshev, inverse Chebyshev, and elliptical filter modules with those generated by MATLAB. The frequency response for the bessel filter module is also given. However, there is no implementation of a digital bessel filter in MATLAB. In all cases, the responses show a steeper rolloff for the higher order filters. Both coefficients computed by the filter modules and by MATLAB's functions are identical for filter orders up to 15. In cases of more than 16

coefficients, the accuracy of the frequency response yielded by the coefficients starts to decrease. This limitation may be caused by the rounding off of floating point number variables with the large number of multiplication and addition operations used to compute the filter coefficients.

### 6.2.2 FIR filter module verification

For FIR filters, a higher order filter corresponds to a higher number of coefficients. From Appendix A, the three FIR filter design methods show that as the number of coefficients increases, the transition between the passband and stopband is much steeper, as expected. To illustrate this characteristic, an error curve is plotted showing the difference between the frequency response of an ideal low-pass filter and the frequency response of the reconfigurable filter for each filter design method. An example of an error curve for the “Frequency Sampling Design” method with  $N=10$  and  $N=30$  coefficients is shown in Figure 43.



**Figure 43** Error curve for reconfigurable filter using the “Frequency Sampling Design” method. (a) for  $N=10$  coefficients. (b) for  $N=30$  coefficients.

The spike in the error curve corresponds to the transition between the passband and stopband. It is noted that for  $N=30$ , the spike is more compressed along the x-axis than for the  $N=10$  case, indicating a faster transition between the passband and stopband.

As with the IIR filter modules, the frequency responses of the reconfigurable FIR filters are also presented in Appendix A. The power spectrum for each of these filters is also shown. In all cases, the FIR filters are designed to have a normalized cutoff frequency of 0.5.

The FIR coefficients computation modules that use the “Design by Windowing” method and the “Parks-McClellan” method have been verified with MATLAB’s respective functions and are shown in Appendix A. It is shown that the reconfigurable filter modules yield the same general shape of frequency responses as their respective counterpart in MATLAB. The FIR coefficients computation modules for the “Frequency Sampling Design” method, the Gaussian filter coefficients computation module and the digital integrator have not been verified with MATLAB’s toolbox because the corresponding functions in MATLAB were not available.

As shown in the error curves, the “Frequency Sampling Design” method yields frequency responses for which the transition between passband and stopband becomes steeper as the filter order increases. A consequence of the steeper transition is an overshoot of the frequency response at the start of the transition which measured approximately 11% of the passband magnitude. The attenuation in the stopband is shown to start at  $-15\text{dB}$  and gradually rolls off to approximately  $-30\text{dB}$ .

The “Design by Windowing” method yields better frequency responses with less ripple in the stopband and in the passband, and smaller overshoot at the transition. Furthermore, the results indicated significantly better attenuation in the stopband at around  $-50\text{dB}$ . However, when observing the error curves in Table A10 of Appendix A, a wider spike suggests a more gradual transition between the passband and stopband.

The frequency responses of the “Parks-McClellan” filters exhibited ripples in both the passband and stopband as did the “Frequency Sampling Design” method. A difference in the amplitude of the frequency responses between the reconfigurable filter module and the MATLAB functions can be attributed to different values of the passband and stopband ripple. With a greater stopband ripple allowed in the MATLAB case, less attenuation is noted in the power spectrum when compared with the reconfigurable filter modules. The error curves for the Parks-McClellan method demonstrated transitions between passband and stopband that were comparable to the “Frequency Sampling Design” method. While the Parks-McClellan method provides flexibility in setting the passband and stopband ripple, it is much more computationally intensive. This may have significant implications when choosing a filter design method for real-time processing requirements.

Frequency response and power spectrum curves are also plotted for a Gaussian filter with  $BT=0.2$  for  $N=10, 20$ , and  $30$  coefficients. As mentioned in Section 4.3.2.2, the “Frequency Sampling Design” method was used to implement the reconfigurable Gaussian filter module.

The results showed that while the main lobe did not change significantly as the number of coefficients increased, more attenuation in the stopband was observed as  $N$  increased.

Results for the digital integrator module are also shown in Appendix A. The output curve shows the integration of a bipolar input digital bit stream.

## 7 SUMMARY

This report describes a project to develop reconfigurable IIR and FIR filter modules for use in the ROBR project. The theory needed to understand digital signals, LTI systems and filters was introduced in Section 2. The concept of IIR and FIR filters was presented in Section 3 and Section 4. Both the IIR and FIR implementations were represented using the Direct Form structure. Equations for the analog frequency response of various types of IIR filters were presented. Examples include Butterworth, Chebyshev, and Elliptical filters.

The design methods used to implement IIR filters produce a discrete filter design from analog design prototypes. Two methods have been used to implement IIR filters: the Bilinear transformation and the Impulse Invariance method. These two methods perform a transformation on an analog design to obtain a discrete design.

Methods used to produce FIR filters involve sampling, IDFT computation and an optimization algorithm. Three design methods have been used to implement FIR filter modules: the "Frequency Sampling Design" method, the "Design by Windowing" method and the "Parks-McClellan" method. The "Frequency Sampling Design" method involves two steps in the computation of the filter coefficients: building an ideal frequency response vector, and computing an IDFT. A vector is a discrete sequence of elements. Building the vector is equivalent to sampling the ideal frequency response. The "Design by Windowing" method involves the same steps as the "Frequency Sampling Design" method except that the result of the IDFT performed on the desired frequency response samples are subsequently multiplied by a vector containing amplitude samples of a window function. A Hamming window has been used for this implementation. The "Parks-McClellan" method is based on the Alternation Theorem from optimization theory. The Remez Exchange algorithm is used to find the optimal set of extremal frequencies. The goal of the method is to compute the coefficients for the best approximation of a desired frequency response. The Remez exchange algorithm is a set of conditional statements that, when applied, produce an optimal frequency response. The design of a Gaussian filter using the "Frequency Sampling Design" method is also presented.

Reconfigurable filter modules were implemented for five types of IIR filters and four types of FIR filters. The implemented types of IIR filters are the Butterworth filter, the Chebyshev filter, the Inverse Chebyshev filter, the Elliptical filter and the Bessel filter. Two FIR filter modules, based on the "Frequency Sampling Design" method, were implemented called *freqsampling*, and *gauss*, the latter of which is the Gaussian filter module implementation. An FIR filter module was also implemented using the "Design by Windowing" method and is called *hamming*. The "Parks-McClellan" FIR implementation is called *remezex*.

Coefficients generated by the reconfigurable filter modules were used to process an input impulse function. The resulting frequency response was compared to that generated by MATLAB and is presented in Appendix A. The digital reconfigurable IIR filter modules were found to yield the same responses as the corresponding functions in MATLAB. The FIR digital filter modules also yielded the same frequency responses as MATLAB. The Parks-McClellan method provides the flexibility to adjust the passband and stopband ripple

while yielding a comparatively steep transition. However, the algorithm is much more computationally intensive. A Gaussian filter module and a digital integrator module were also successfully implemented. The Gaussian filter and digital integrator modules will facilitate the development of a GMSK modulator for the ROBR.

The "Design by Windowing" and "Parks-McClellan" filter modules were successfully adapted for use on a DSP board. As well, a module for processing a signal using the FIR filter coefficients generated was implemented on the DSP. The DSP board used was the Daytona Dual c62 processor board from Spectrum Signal Processing Inc. Issues related to dynamic memory allocation and handshaking between the host and the DSP were discussed. Further work is required to completely adapt the filter modules for the ROBR, to assess the ability to reconfigure the modules while the ROBR is operating, and to resolve any time critical issues for the ROBR where more computationally intensive algorithms are used.

## 8 REFERENCES

- [1] Allan V. Oppenheim and Ronald W. Schaffer, *Discrete-Time Signal Processing*, Prentice Hall Signal, Englewood Cliffs, New Jersey
- [2] Micheal J. Corinthios, *Analyse des signaux, Quatrième édition*, Département de génie électrique et de génie informatique, École Polytechnique de Montréal, Janvier 2000.
- [3] T.W. Parks and C. S. Burrus, *Digital Filter Design*, John Wiley & Sons, Inc.
- [4] Alexander D. Poularikas, *The Handbook of Formulas and Tables for Signal Processing*, IEEE Press and CRC Press, 1999.
- [5] C. Britton Rorabaugh, *Digital Filter Designer's Handbook, Second Edition*, McGraw-Hill
- [6] A. Antonious, "Accelerated procedure for the design of equiripple nonrecursive, digital filters" IEE Proc., Pt. G, vol. 129, pp. 1-10, Feb 1982 (see IEEE Proc., Pt. G vol. 129, p.107 June 1982 for errata)
- [7] A. Antonious, "New improved method for the design of weighted-Chebyshev, nonrecursive, digital filters" IEEE Trans. Circuits Syst., vol. CAS-30, pp. 740-750, Oct. 1983.
- [8] Andreas Antonious, *Digital Filters: Analysis and Design*, McGraw-Hill Book Company
- [9] Daytona Dual 'C6x PCI Board Technical Reference, Document Number 500-00383, Revision 2.00, May 1999
- [10] Daytona/Barcelona 'C6x PCI Board Windows NT Programming Guide, Document Number 500-00384, Revision 1.10, May 1999
- [11] TMS320C62x/C67x, Programmer's Guide, Literature Number: SPRU198B, Texas Instruments, February 1998
- [12] Kazuaki Murota and Kenkichi Hirade, "GMSK Modulation for Digital Mobile Radio Telephony", IEEE Transactions on Communications, Vol. COM-29, NO. 7, JULY 1981

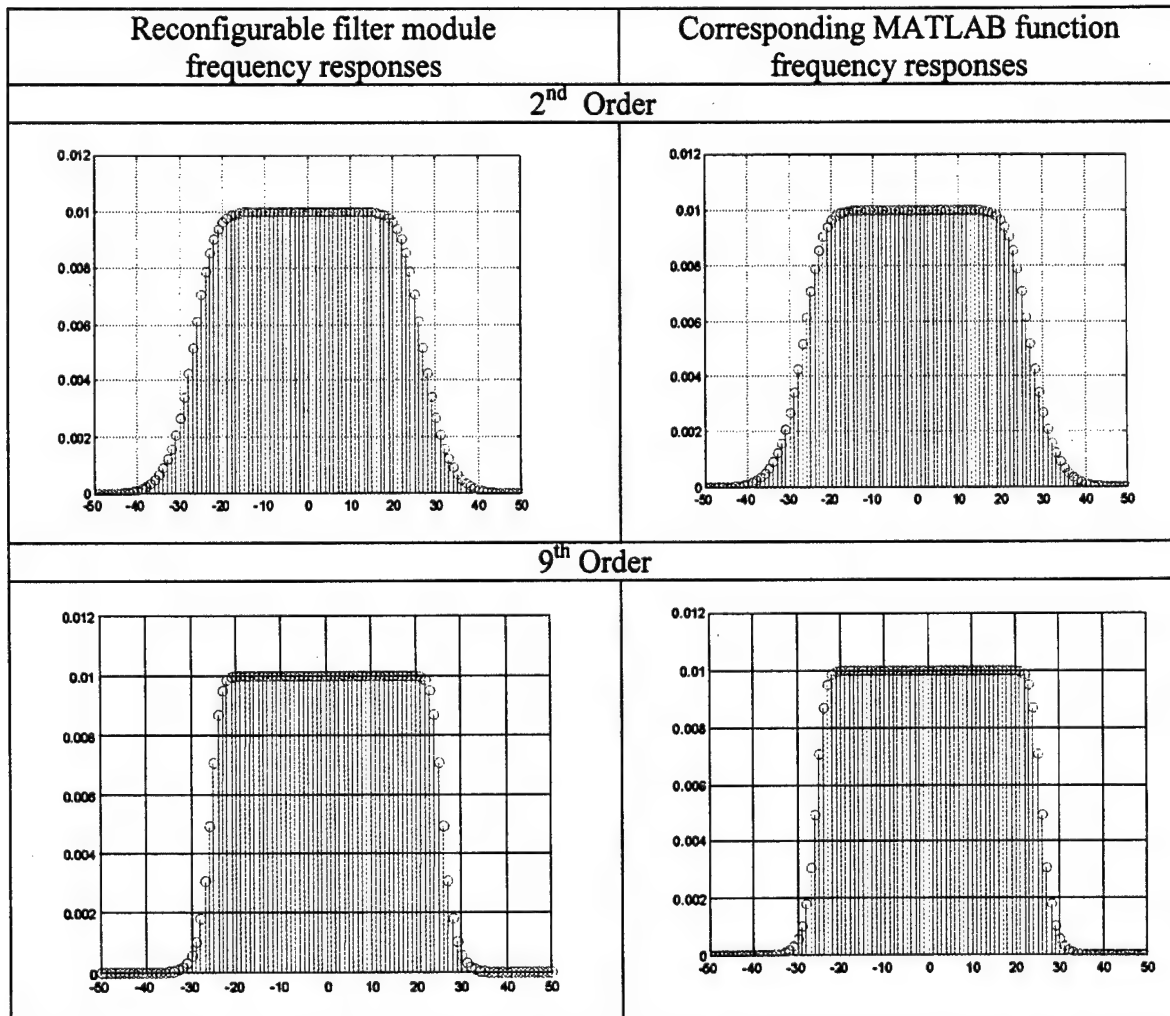


## APPENDIX A

## A1. RESULTS

### A1.1. IIR filter modules

#### A1.1.1. Butterworth filter



**Table A1 Results for the Butterworth filter module**

A1.1.2. Chebyshev filter

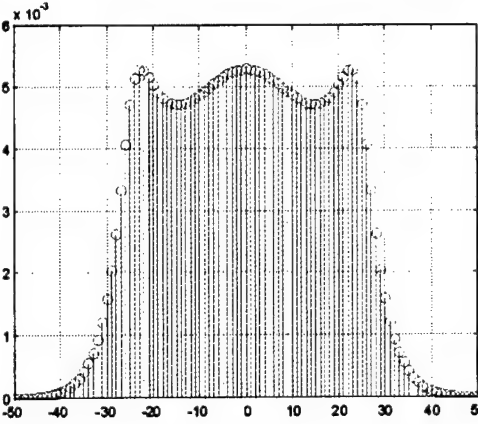
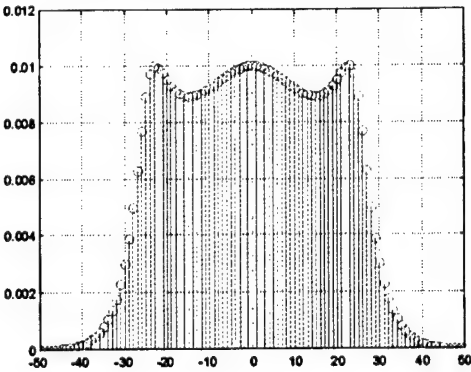
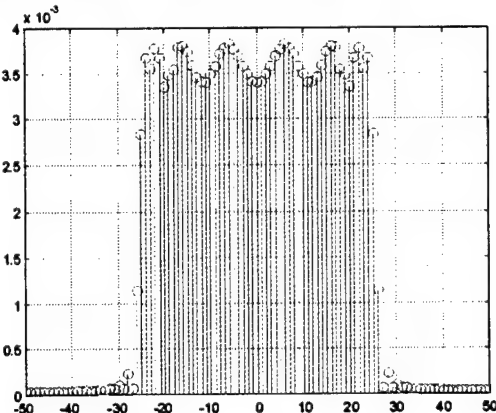
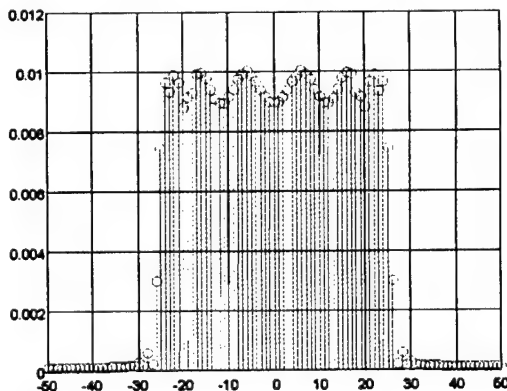
Reconfigurable filter module frequency responses	Corresponding MATLAB function frequency responses
2 <sup>nd</sup> Order	
	
9 <sup>th</sup> Order	
	

Table A2 Results for the Chebyshev filter module

A1.1.3. Inverse Chebyshev filter

Reconfigurable filter module frequency responses	Corresponding MATLAB function frequency responses
2 <sup>nd</sup> Order	
9 <sup>th</sup> Order	

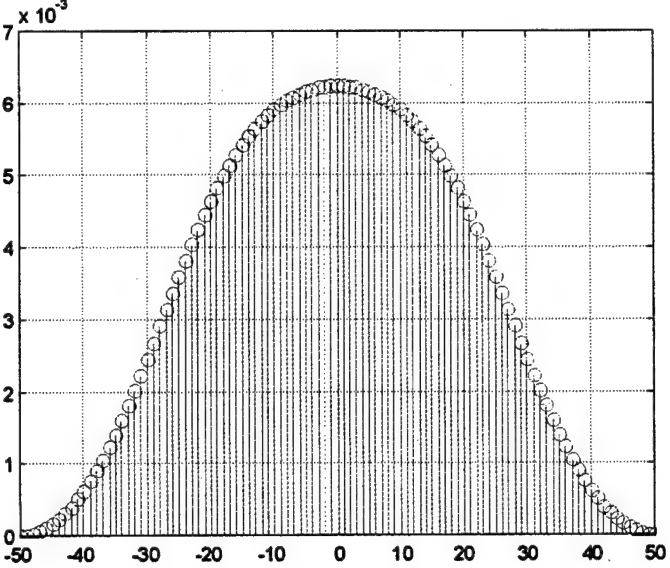
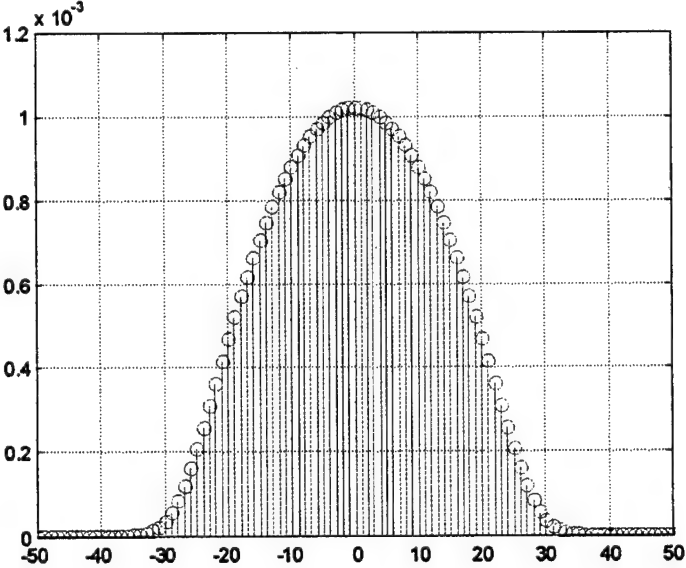
Table A3 Results for the Inverse Chebyshev filter module

A1.1.4. Elliptical filter

Reconfigurable filter module frequency responses	Corresponding MATLAB function frequency responses
2 <sup>nd</sup> Order	
9 <sup>th</sup> Order	

Table A4 Results for the Elliptical filter module

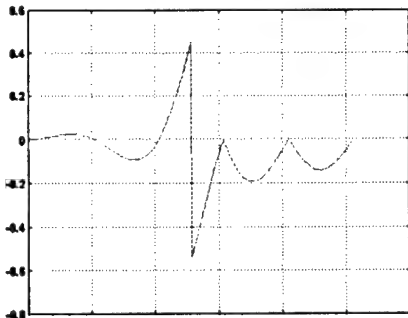
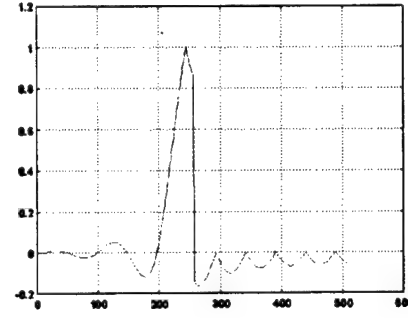
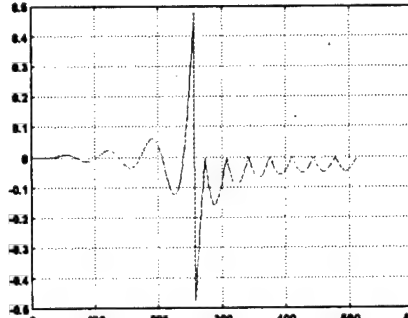
**A1.1.5. Bessel filter**

Reconfigurable filter module Frequency responses		Corresponding MATLAB function frequency responses
2 <sup>nd</sup> Order		
		No corresponding MATLAB function available
9 <sup>th</sup> Order		
		No corresponding MATLAB function available

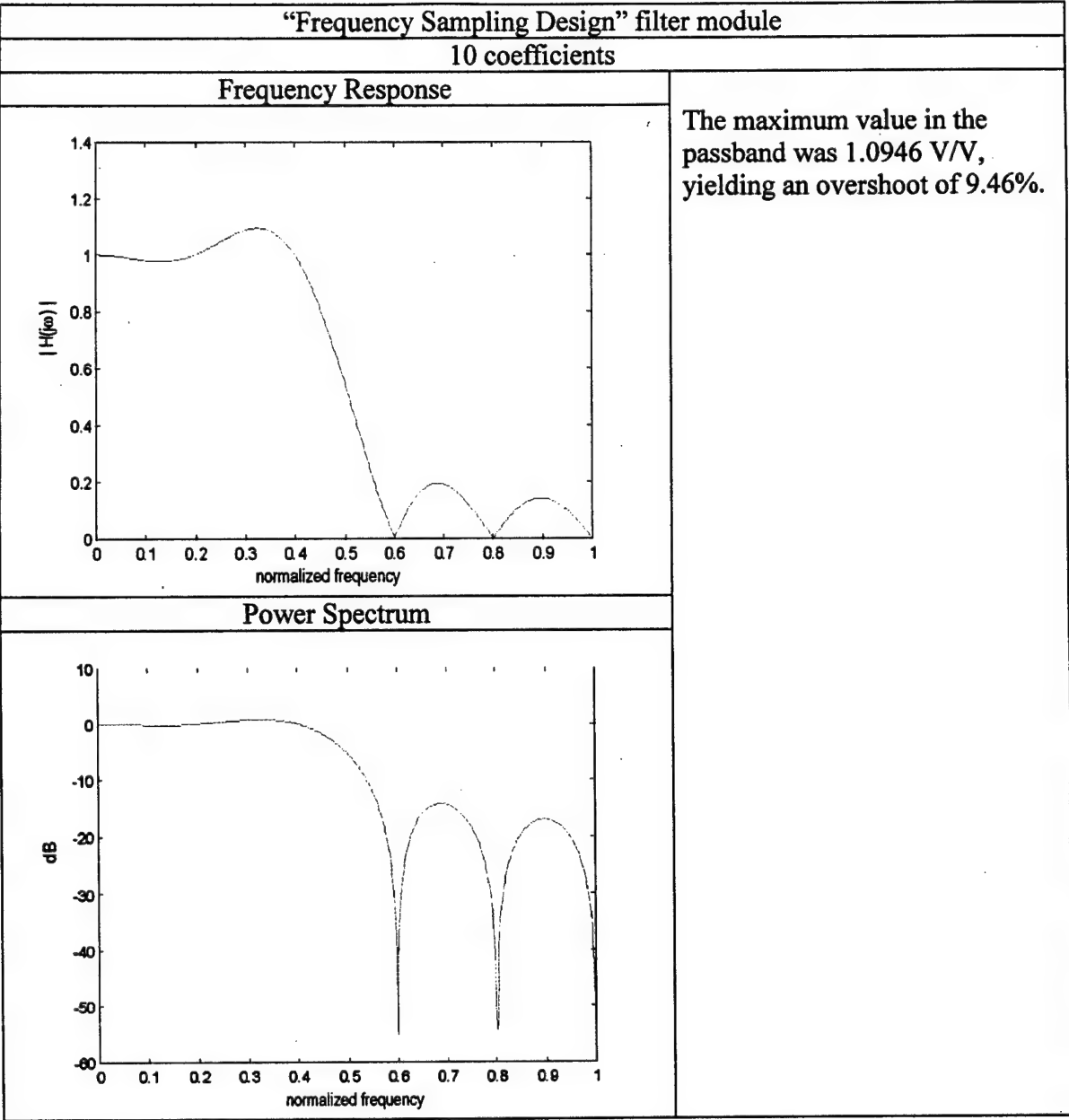
**Table A5 Results for the Bessel filter module**

## A1.2. FIR filtering modules

### A1.2.1. Frequency sampling design method

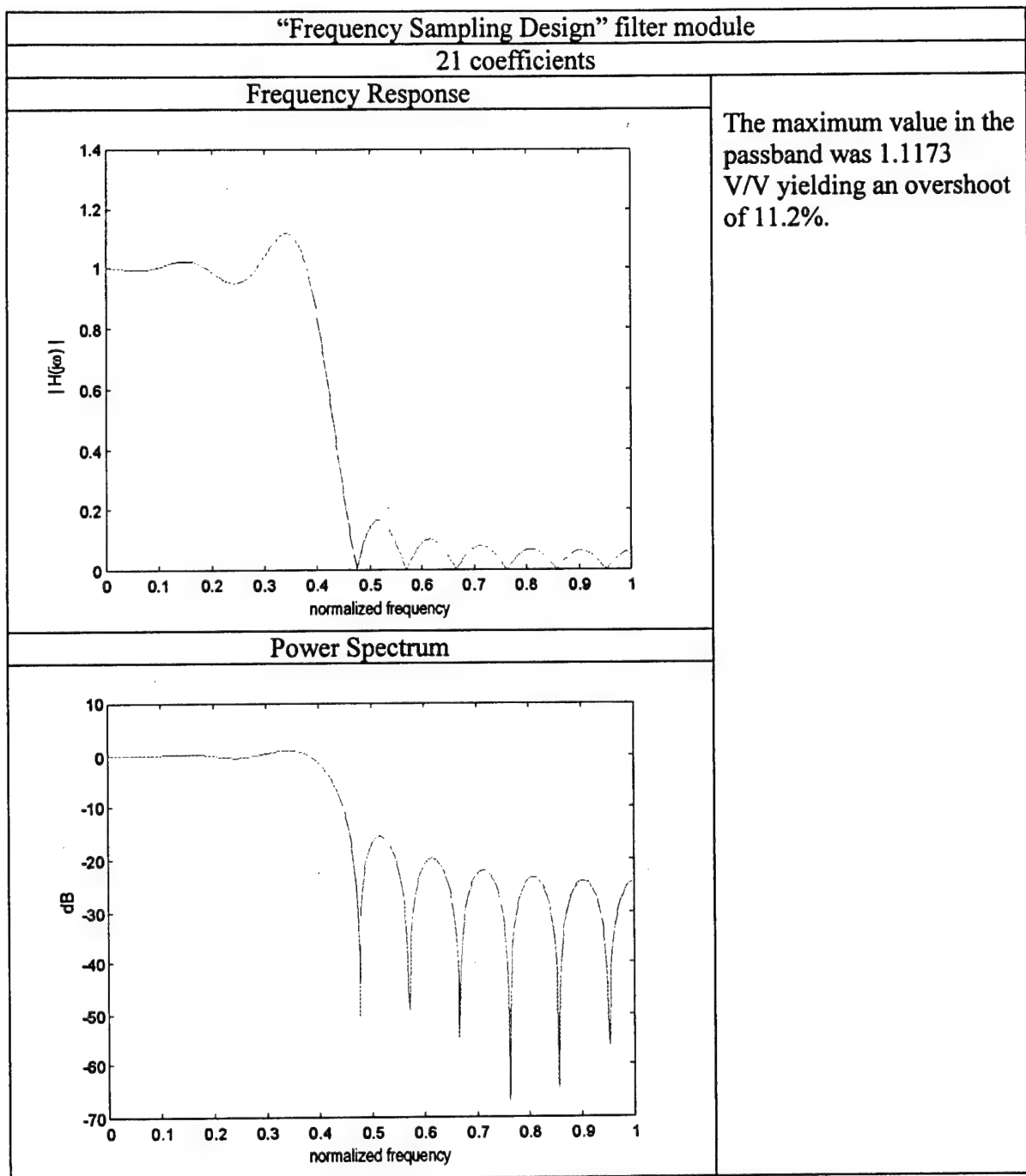
Reconfigurable filter module Coefficients	Error Curve (Ideal low-pass filter - Reconfigurable filter) vs. samples	Mean error
0.071592 -0.079360 -0.100000 0.155754 0.452015 0.452015 0.155754 -0.100000 -0.079360 0.071592		-0.0572
0.037334      0.311490 -0.021192      0.070096 -0.049873      -0.085807 -0.000000      -0.066090 0.059380      0.030376 0.030376      0.059380 -0.066090      -0.000000 -0.085807      -0.049873 0.070096      -0.021192 0.311490      0.037334 0.428571		0.0354
0.450364      -0.023603 0.150672      0.023864 -0.091068      0.024402 -0.065771      -0.025247 0.051918      -0.026453 0.043277      0.028104 -0.037453      0.030329 -0.033333      -0.033333 0.030329      -0.037453 0.028104      0.043277 -0.026453      0.051918 -0.025247      -0.065771 0.024402      -0.091068 0.023864      0.150672 -0.023603      0.450364		0.0265

**Table A6 Results for the Frequency Sampling Design filter module**

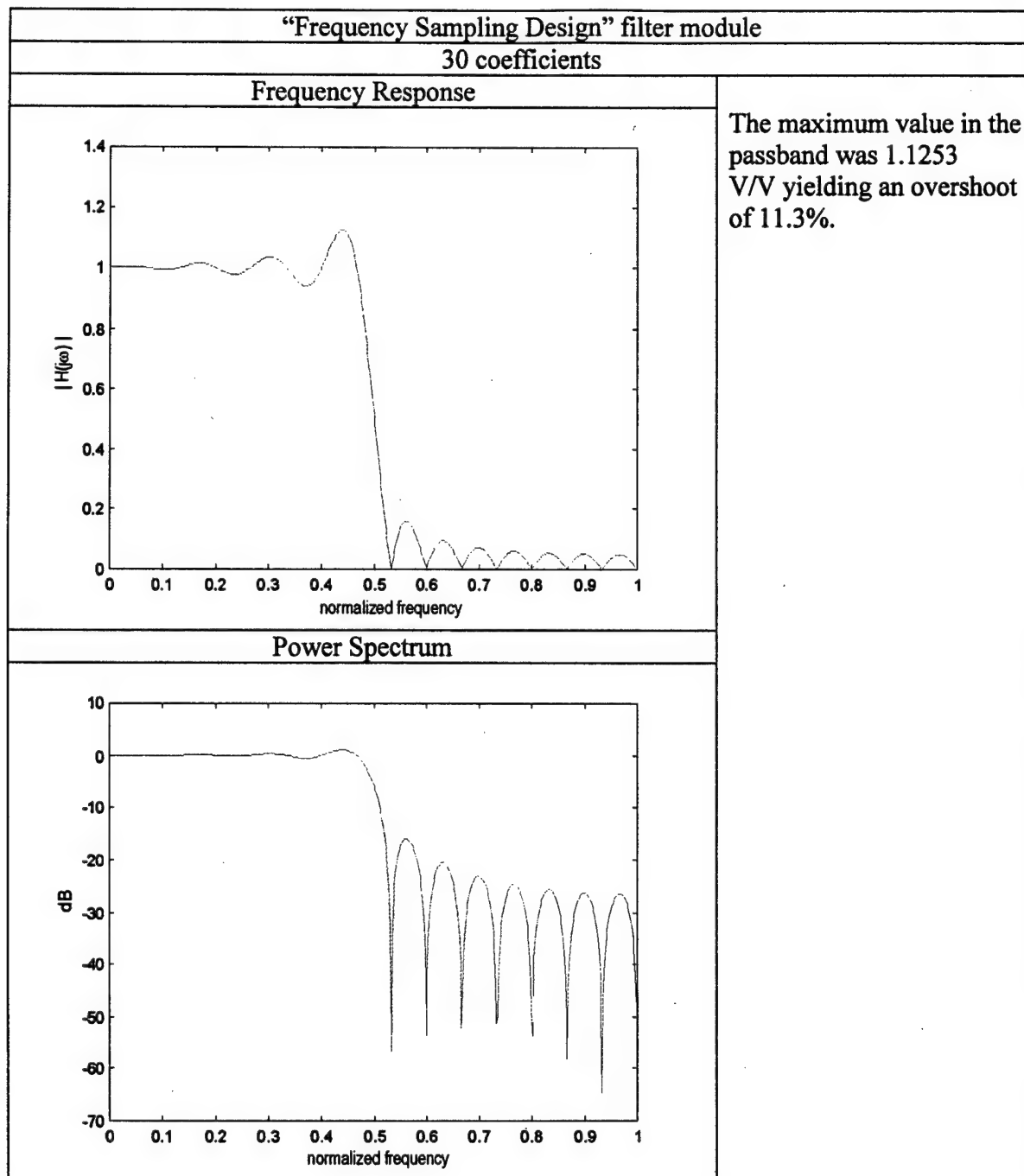


**Table A7 Frequency and power spectrum for N=10 FIR filter using the Frequency Sampling Design filter module**



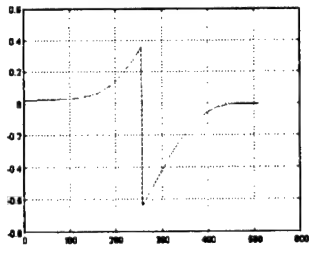
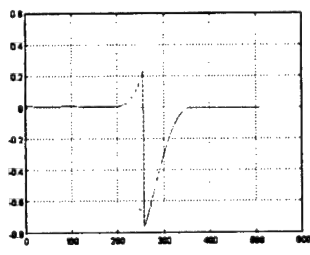
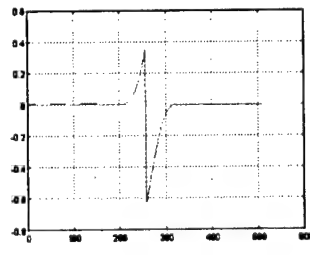


**Table A8 Frequency and power spectrum for N=21 FIR filter using the Frequency Sampling Design filter module**

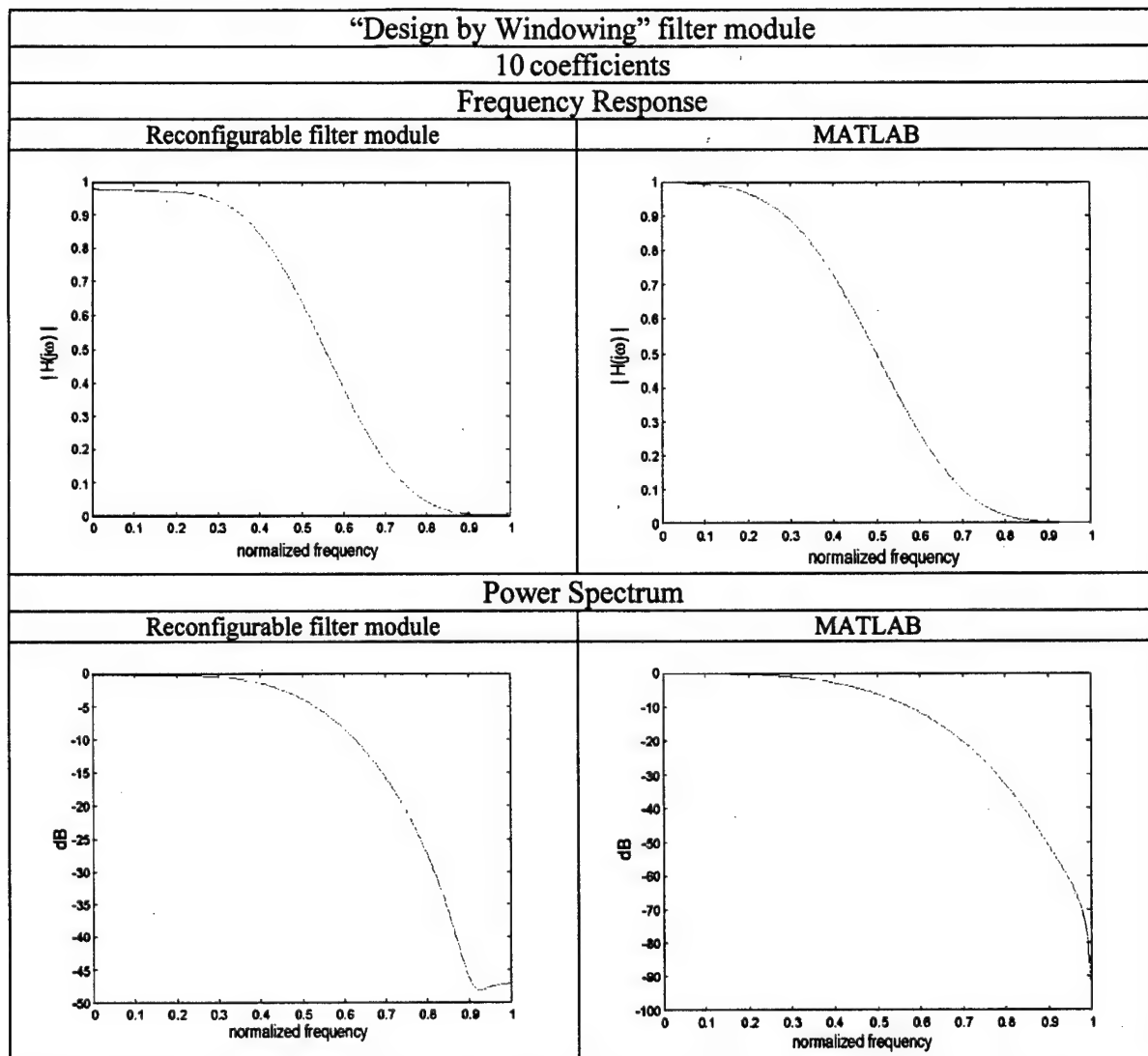


**Table A9 Frequency and power spectrum for N=30 FIR filter using the Frequency Sampling Design filter module**

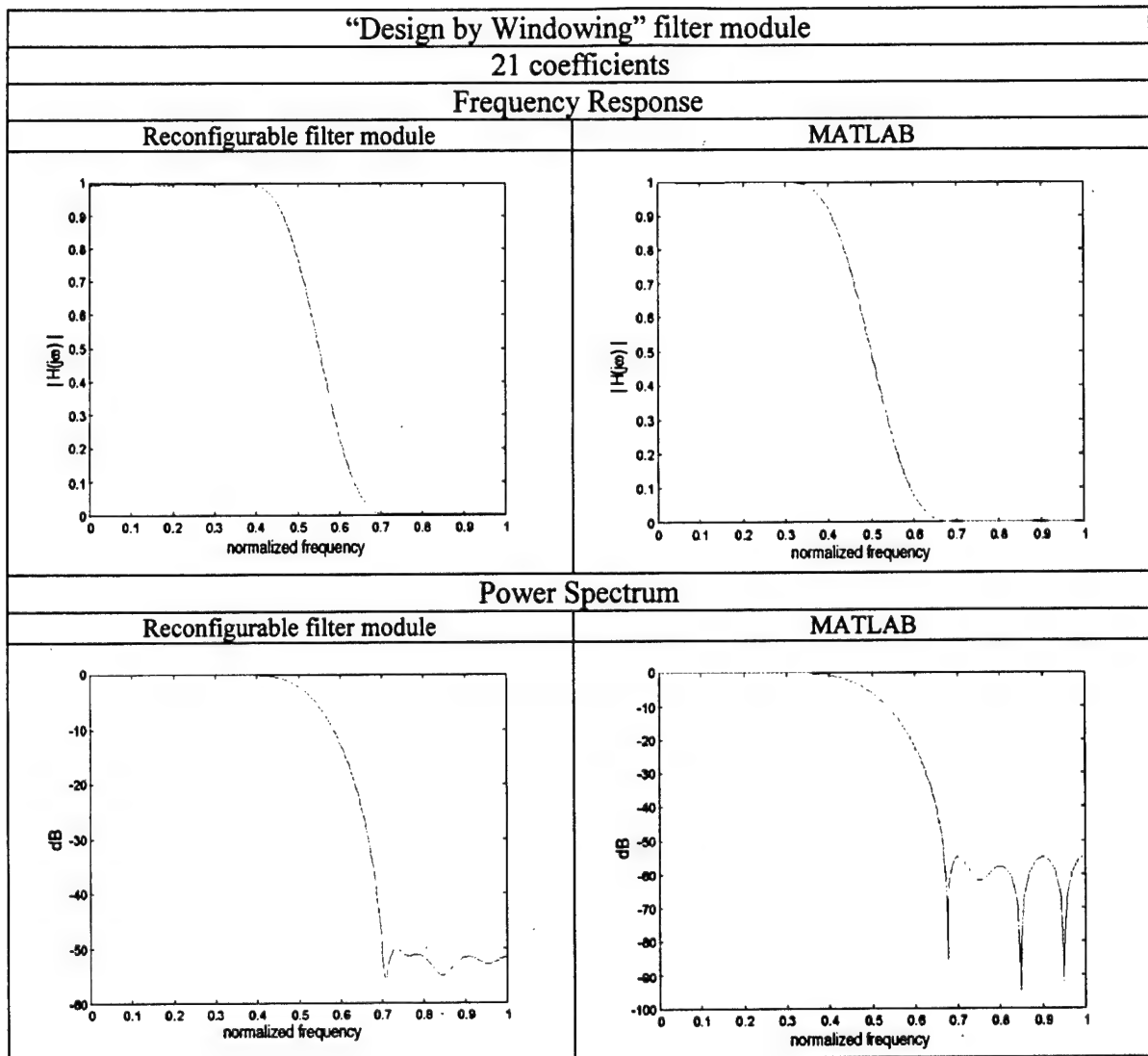
### A1.2.2. Design by windowing with a Hamming window

Filtering modules coefficients		Mean error	Error Curve (Ideal low-pass filter - Reconfigurable filter) vs. samples	MATLAB generated coefficients		Mean error
N=10	0.005802 -0.020847 -0.027203 0.246347 0.540144 0.311056 -0.045523 -0.051125 0.013607 0.005802	0.0442		0.0040 -0.0120 -0.0411 0.1147 0.4344 0.4344 0.1147 -0.0411 -0.0120 0.0040		0.0042
N=21	-0.002606 0.109391 0.004495 -0.110106 0.003476 -0.018103 -0.015374 0.052356 -0.002053 -0.002786 0.041446 -0.022686 -0.015239 0.005584 -0.097822 0.007359 0.102079 -0.003339 0.473677 -0.002606 0.484587	-0.0472		0.0000 0.3106 0.0036 0.0000 -0.0000 -0.0858 -0.0122 -0.0000 0.0000 0.0343 0.0343 0.0000 -0.0000 -0.0122 -0.0858 -0.0000 0.0000 0.0036 0.3106 0.0000 0.4991		0.0014
N=30	-0.001900 0.317607 0.002417 -0.016925 0.002653 -0.099554 -0.005340 0.015429 -0.004874 0.051438 0.011946 -0.012922 0.007921 -0.028620 -0.023994 0.009785 -0.011179 0.015347 0.045774 -0.006530 0.014091 -0.007441 -0.093139 0.003764 -0.016201 0.003263 0.310797 -0.002155 0.515847 -0.001900	-0.0159		-0.0012 0.4490 0.0015 0.1465 0.0022 -0.0841 -0.0034 -0.0562 -0.0052 0.0399 0.0077 0.0291 0.0110 -0.0213 -0.0155 -0.0155 -0.0213 0.0110 0.0291 0.0077 0.0399 -0.0052 -0.0562 -0.0034 -0.0841 0.0022 0.1465 0.0015 0.4490 -0.0012		2.7510e-004

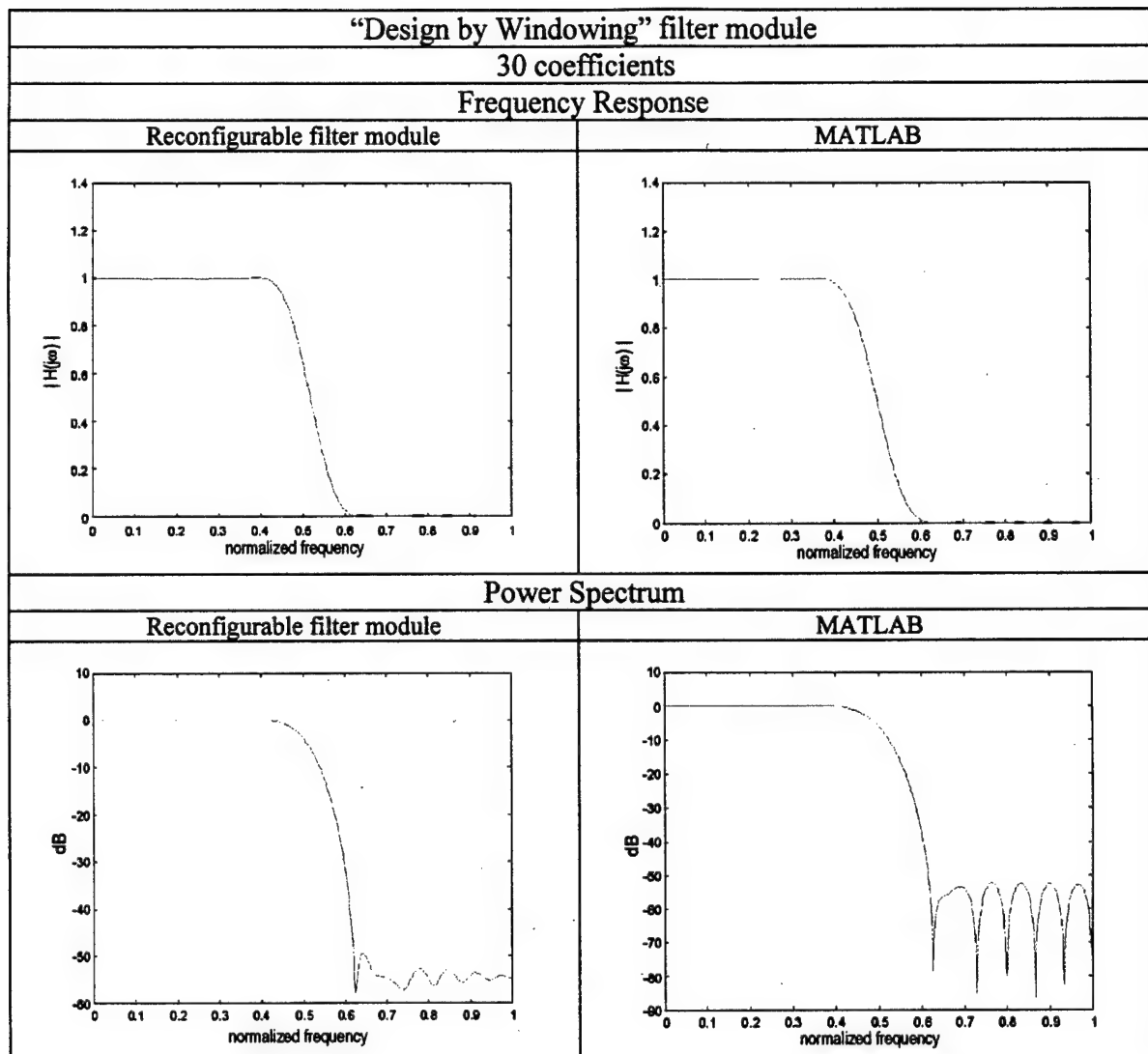
**Table A10 Results for the Design by Windowing filter module**



**Table A11 Frequency and power spectrum for N=10 FIR filter using the Design by Windowing filter module**

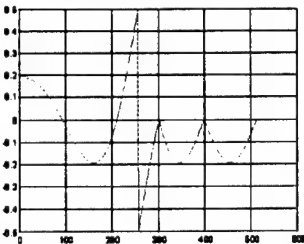
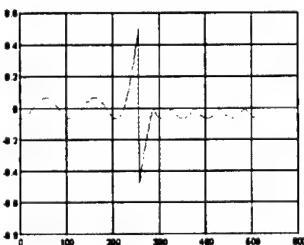
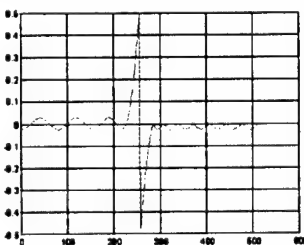


**Table A12 Frequency and power spectrum for N=21 FIR filter using the Design by Windowing filter module**

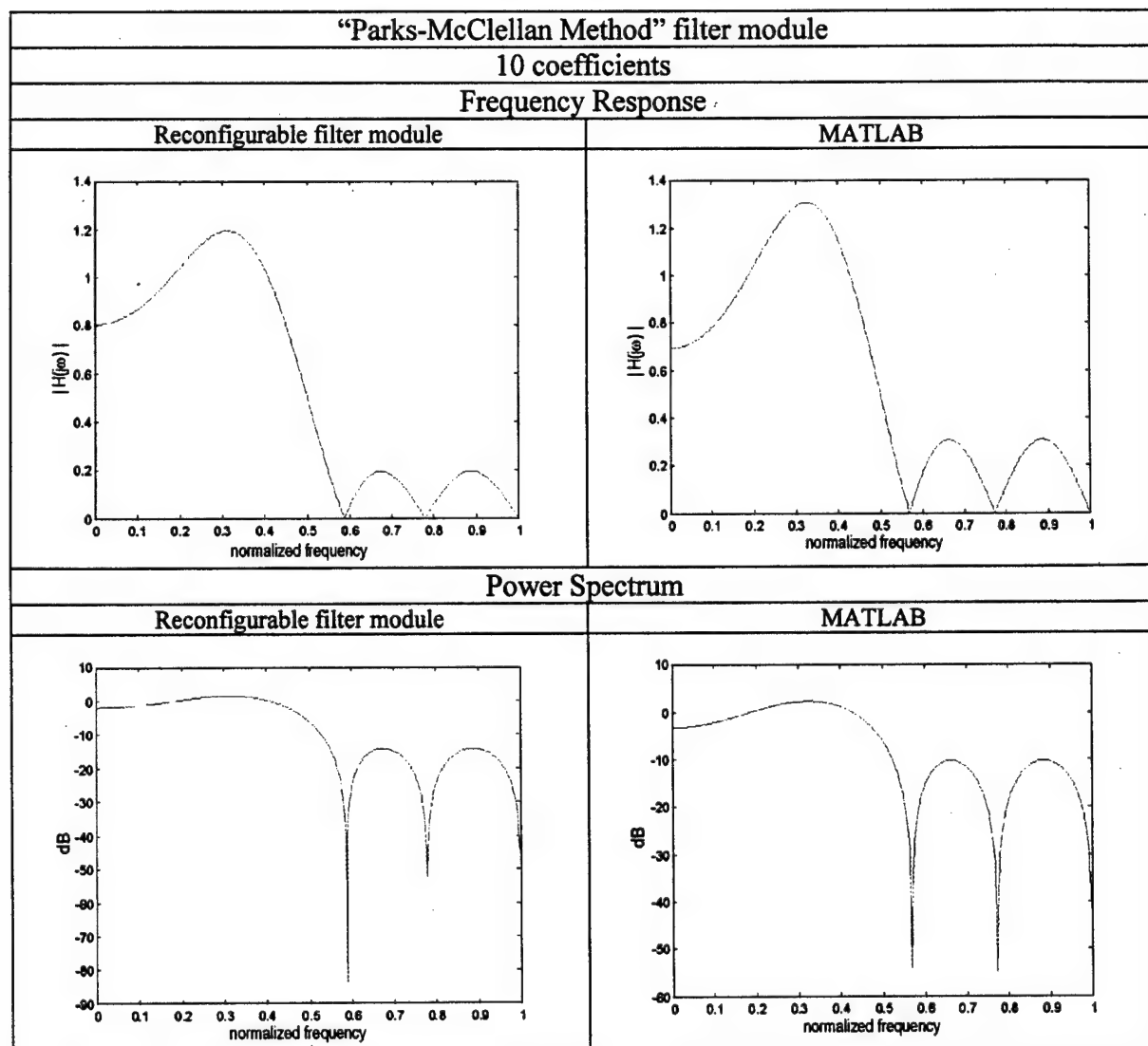


**Table A13 Frequency and power spectrum for N=30 FIR filter using the Design by Windowing filter module**

### A1.2.3. Parks-McClellan method

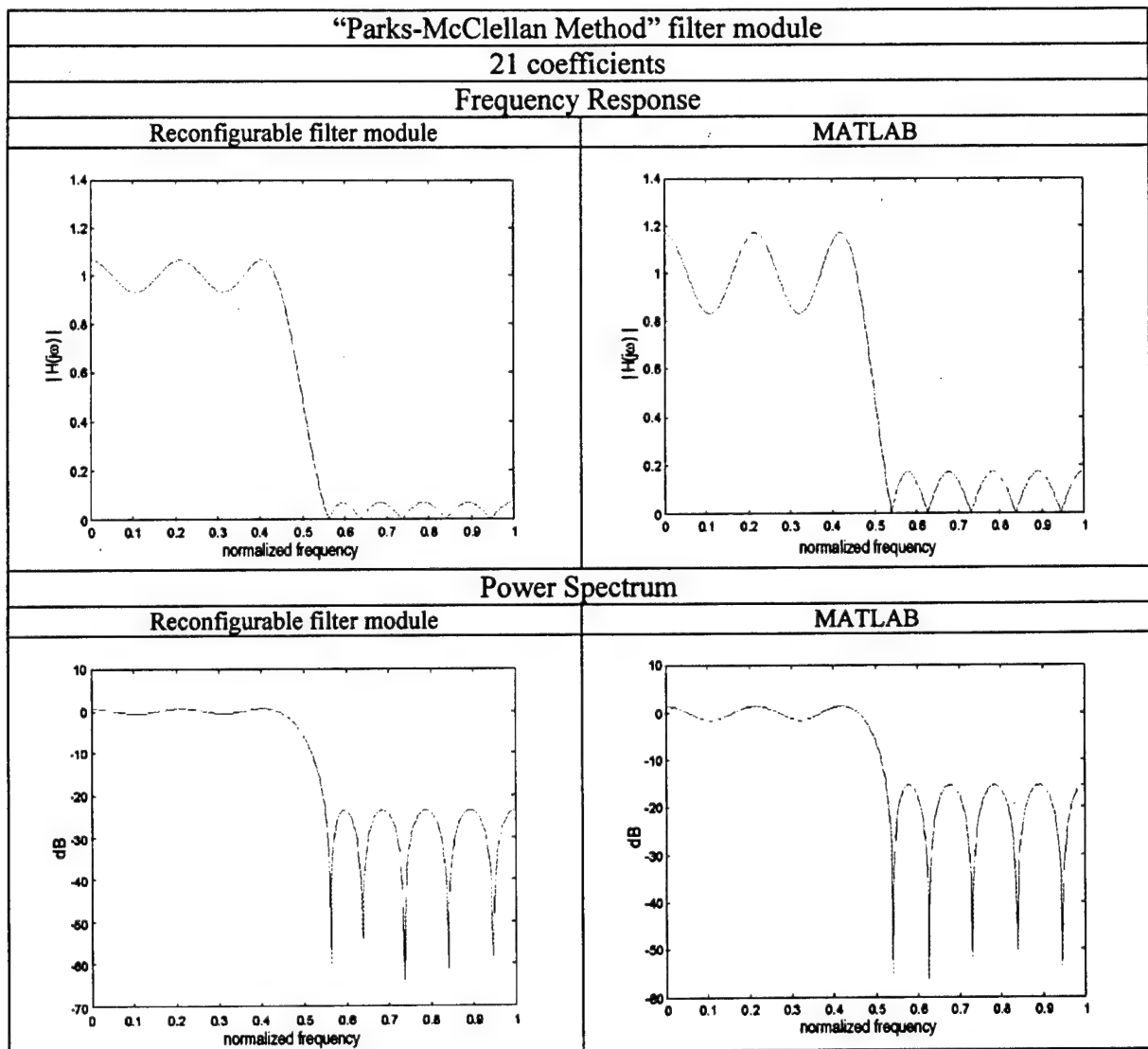
Filtering modules coefficients			Mean error	Error Curve (Ideal low-pass filter - Reconfigurable filter) vs. samples	MATLAB generated coefficients		Mean error
N=10	0.059964 -0.125401 -0.115243 0.138672 0.444618 0.444618 0.138672 -0.115243 -0.125401 0.059964		-0.0560		0.0842 -0.1751 -0.1373 0.1335 0.4412 0.4412 0.1335 -0.1373 -0.1751 0.0842		-0.0932
N=21	0.000017 0.048051 -0.000023 -0.036911 -0.000015 0.057263 0.057263 -0.000015 0.000001 -0.036911 -0.102173 -0.000023 0.000012 0.048051 0.316962 0.500019	0.316962 0.000012 -0.102173 0.000001 0.057263 -0.000015 -0.036911 -0.000023 0.000012 0.048051 0.000017	-0.0188		-0.0001 0.1045 0.0000 -0.0445 -0.0000 0.0628 0.0628 -0.0000 0.0000 -0.1056 0.0001 0.1045 0.3181 0.4999	0.3181 0.0001 -0.1056 0.0000 0.0628 -0.0000 -0.0445 0.0000 0.1045 -0.0001	-0.0530
N=30	-0.008176 0.016396 0.013294 -0.010301 -0.013590 0.017082 0.020431 -0.024314 -0.029749 0.036631 0.046579 -0.061479 -0.088117 0.148791 0.449824	0.449824 0.148791 -0.088117 -0.061479 0.046579 0.036631 -0.029749 -0.024314 0.020431 0.017082 -0.013590 -0.010301 0.013294 0.016396 -0.008176	-0.0069		-0.0257 0.0515 0.0310 -0.0107 -0.0165 0.0227 0.0250 -0.0281 -0.0331 0.0397 0.0490 -0.0635 -0.0895 0.1497 0.4501	0.4501 0.1497 -0.0895 -0.0635 0.0490 0.0397 -0.0331 -0.0281 0.0250 0.0227 -0.0165 -0.0107 0.0310 0.0515	-0.0312

**Table A14 Results for the Parks-McClellan design filter module**

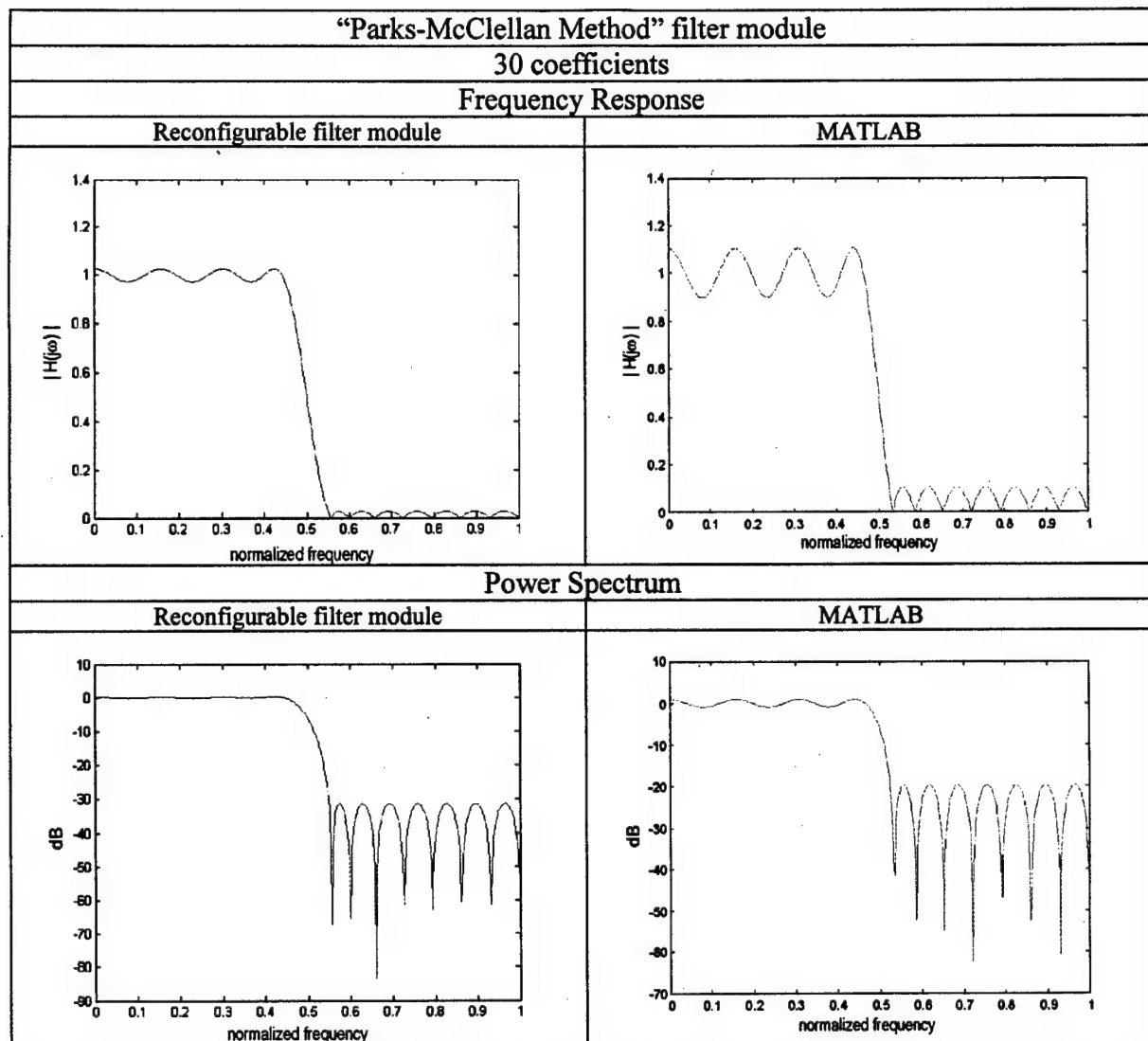


**Table A15 Frequency and power spectrum for N=10 FIR filter using the Parks-McClellan filter module**





**Table A16 Frequency and power spectrum for N=21 FIR filter using the Parks-McClellan filter module**



**Table A17 Frequency and power spectrum for N=30 FIR filter using the Parks-McClellan filter module**

#### A1.2.4. Gaussian filter

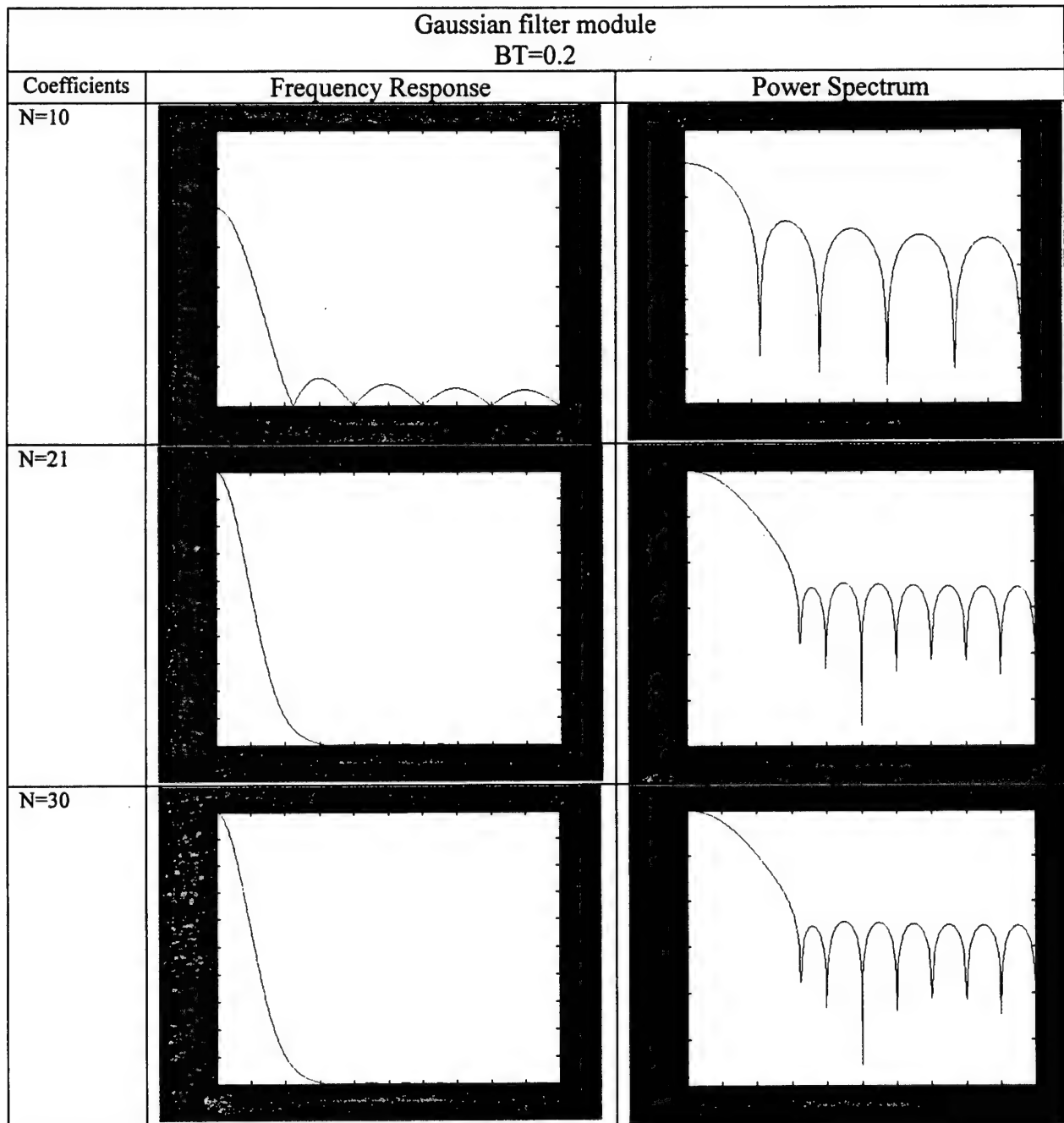


Table A18 Frequency and power spectrum results for the Gaussian filter module

### A1.2.5. Digital Integrator

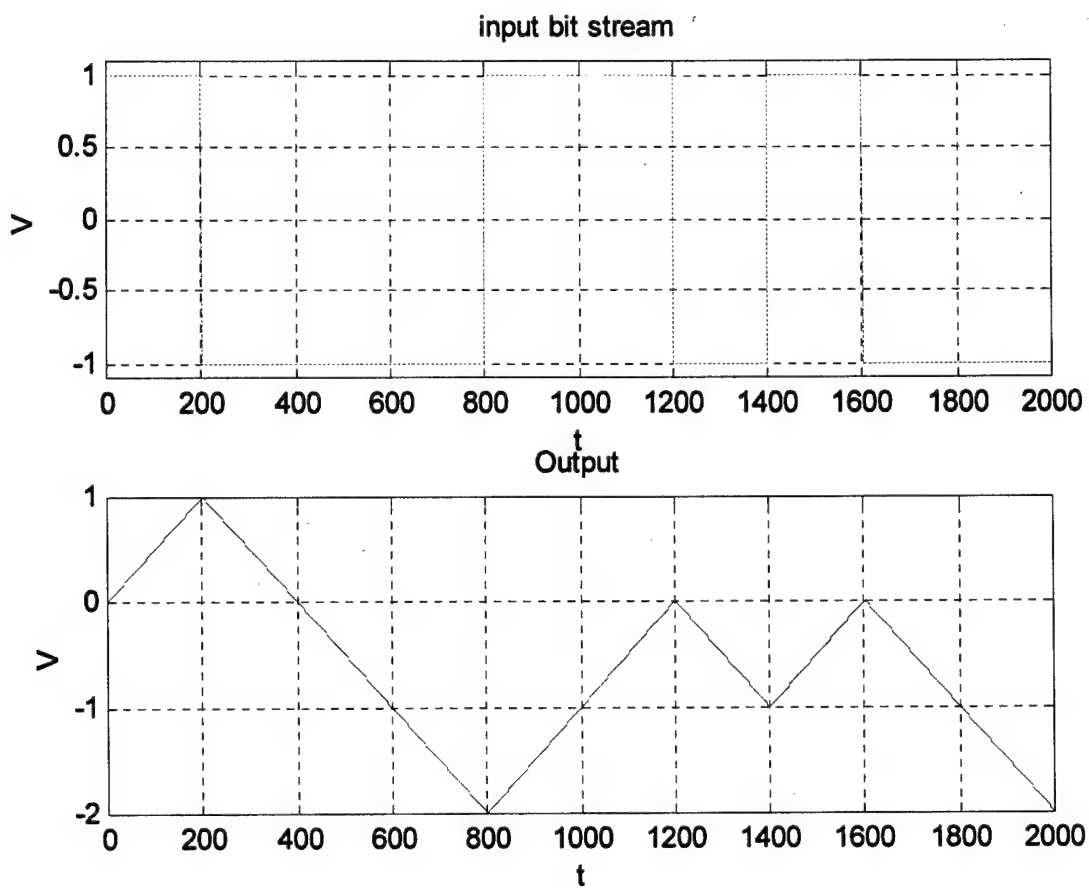


Table A19 Results for the digital integrator filter module

## APPENDIX B

## Steps to calculate poles and zeros of the elliptic filter

The Elliptical filter coefficients computation module implements the following steps to calculate the poles and zeros of such a filter. Poles and zeros are then used to compute the coefficients of the filter [4].

$\omega_p$  = passband frequency

$\omega_s$  = stopband frequency

$A_p$  = maximum passband loss (dB)

$A_s$  = maximum stopband loss (dB)

$k$  = selectivity factor =  $\omega_p / \omega_s$

$$1. k' = \sqrt{1 - k^2}$$

$$2. q_0 = \frac{1}{2} \left( \frac{1 - \sqrt{k'}}{1 + \sqrt{k'}} \right)$$

$$3. q = q_0 + 2q_0^5 + 15q_0^9 + 150q_0^{13}$$

$$4. D = \frac{10^{0.1A_s} - 1}{10^{0.1A_p} - 1}$$

$$5. n \geq \frac{\log 16D}{\log(1/q)}$$

$$6. \Lambda = \frac{1}{2n} \ln \frac{10^{0.05A_p} + 1}{10^{0.05A_p} - 1}$$

$$7. \sigma_0 = \left| \frac{2q^{1/4} \sum_{m=0}^{\infty} (-1)^m q^{m(m+1)} \sinh[(2m+1)\Lambda]}{1 + 2 \sum_{m=1}^{\infty} (-1)^m q^{m^2} \cosh 2m\Lambda} \right|$$

$$8. W = \sqrt{(1 + k\sigma_0^2) \left( 1 + \frac{\sigma_0^2}{k} \right)}$$

$$9. \Omega_i = \frac{2q^{1/4} \sum_{m=0}^{\infty} (-1)^m q^{m(m+1)} \sin \left[ \frac{(2m+1)\pi\mu}{n} \right]}{1 + 2 \sum_{m=1}^{\infty} (-1)^m q^{m^2} \cosh \left[ \frac{2m\pi\mu}{n} \right]}$$

$$\mu = \begin{cases} i & \text{For } n \text{ odd} \\ i - \frac{1}{2} & \text{For } n \text{ even} \end{cases} \quad i = 1, 2, \dots, l$$

$$10. V_i = \sqrt{(1 + k\Omega_i^2) \left( 1 + \frac{\Omega_i^2}{k} \right)}$$

$$11. a_{0i} = \frac{1}{\Omega_i^2}$$

$$b_{0i} = \frac{(\sigma_0 V_i)^2 + (\Omega_i W)^2}{(1 + \sigma_0^2 \Omega_i^2)^2}$$

$$b_{1i} = \frac{2\sigma_0 V_i}{1 + \sigma_0^2 \Omega_i^2}$$

$$12. H_0 = \begin{cases} \sigma_0 \prod_{i=1}^r \frac{b_{0i}}{a_{0i}} & \text{For } n \text{ odd} \\ 10^{-0.05 A_p} \prod_{i=1}^r \frac{b_{0i}}{a_{0i}} & \text{For } n \text{ even} \end{cases}$$

The series in steps 7 and 9 converge rapidly, and three to four terms are sufficient for most purposes. Using the quadratic formula, the  $i^{\text{th}}$  pair of complex pole values can be expressed as

$$p_i = \frac{-b_{1i} \pm \sqrt{b_{1i}^2 - 4b_{0i}}}{2} \quad i = 1, 2, \dots, l$$

The zeros occur at

$$z_i = \pm j\sqrt{a_{0i}} \quad i = 1, 2, \dots, l$$

## APPENDIX C



## Header file, "filter.h"

```
/******
 *Purpose:   This library defines the structure, functions and constants to
 *           implement complex number manipulation
 * Author:   Benoit Gosselin
 * Date:
 *
 *****/

#ifndef FILTER_H
#define FILTER_H

#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#define PI 3.14159265358979
#define EPSILON 1.0e-06

/* This type implements a complex number structure*/

struct cnum
{
    float Q;
    float I;
};

/******
 *
 * Purpose: This function computes the real part and the imaginary
 *           part of a complex number expressed by  $r \cdot \exp(i \cdot \theta)$ 
 * Author:   Benoit Gosselin
 * Date:
 *
 *****/

cnum cfun (float c, float r)
{
    cnum nb;
    nb.Q = r*cos((float)PI*c);
    nb.I = r*sin((float)PI*c);
    return nb;
}

/******
 *
 * Purpose: Generalized form for the previous function to allow
 *           complex input
 * Author:   Benoit Gosselin
 * Date:
 *
 *****/

cnum cexp(cnum c)
{
    cnum nb;
    nb.Q = exp(c.Q)*cos(c.I);
    nb.I = exp(c.Q)*sin(c.I);
    return nb;
}

/******
 *
 * Purpose: Output function. This function prints the coefficients
 *           computed for an IIR filter in a text file.
 *****/
```

```

*
* Format of the text file:
*
*          3.000000          //order of the filter
*          0.040142          //digital static gain
*          //numerator's coeff //denominator's coeff
*          0.040142 0.000000 1.000000 0.000000
*          0.120425 0.000000 -1.057236 0.000000
*          0.120425 0.000000 1.087358 0.000000
*          0.040142 0.000000 -0.708990 0.000000
*
*      the first row is for the real part and the second is for imaginary part
*
* Author:  Benoit Gosselin
* Date:
* parameters:
*
*      cN : coefficients of the numerator
*      cD : coefficients of the denominator
*      N : number of coefficients at the numerator
*      M : number of coefficients at the denominator
*      order : order of the filter
*      Kd : digital static gain of the filter
*      filename : name of the file where should be print the coefficients
*****/

void print_coeff(cnum cN[],cnum cD[],int N,int M,int order,float Kd,char filename[])
{
    int i;
    FILE *outputfile,*matoutputfile;
    outputfile = fopen(filename, "w");
    //print an output file in a different format for Matlab uses
    matoutputfile = fopen("matcoeffout.txt","w");
    fprintf(outputfile,"%f\n", (float)order);
    fprintf(outputfile,"%f\n",Kd);
    cout<<endl<<"numerator's coefficients";

    for( i = 0;i < N; i++)
    {
        cout<<endl<<cN[i].Q<<" + i"<<cN[i].I;
        fprintf(matoutputfile,"%f\n",cN[i].Q);
        fprintf(outputfile,"%f %f %f %f\n",cN[i].Q,cN[i].I,cD[i].Q,cD[i].I);
    }

    for( i = 0;i < N; i++)
        fprintf(matoutputfile,"%f\n",cN[i].I);

    cout<<endl<<"denominator's coefficients";
    for( i = 0;i < M; i++)
    {
        cout<<endl<<cD[i].Q<<" + i"<<cD[i].I;
        fprintf(matoutputfile,"%f\n",cD[i].Q);
    }
    for( i = 0;i < M; i++)
        fprintf(matoutputfile,"%f\n",cD[i].I);

    fclose(outputfile);
    fclose(matoutputfile);
}

```

```

/*****
 *
 * Purpose: To multiply 2 polynomials together
 * Author:  Benoit Gosselin
 *
 *          N and M are the order of polynomials a and b
 * Date:
 *
 *****/

float * polym(float a[],int N,float b[],int M)
{
    int i,j;
    int length;
    length = (N + M + 1);
    i=j=0;

    float *poly = (float *)malloc((length) * sizeof(float));

    for(i=0; i<length;i++)
        poly[i] = 0.0;

    for(i=0;i<N+1;i++)
    {
        for(j=0;j<M+1;j++)
            poly[length-i-j-1]+=a[i]*b[j];
    }
    return poly;
}

/*****
 *
 * Purpose: If the real part or the imaginary part of the complex
 *          number is too small, set it to 0
 * Author:  Benoit Gosselin
 * Date:
 *
 *****/

cnum set_to_zero(cnum a)
{
    if(fabs(a.Q)<EPSILON)
        a.Q = 0.0;
    if(fabs(a.I)<EPSILON)
        a.I = 0.0;
    return a;
}

```

```

/*****
 *
 * Purpose: This function computes the square root of a complex number
 *
 * Author:  Benoit Gosselin
 * Date:
 *
 *****/

cnum sqrtc(cnum a)
{
    float r, theta;

    r = sqrt(a.Q*a.Q + a.I*a.I);

    theta = atan(a.I/a.Q);

    theta = theta / 2.0;

    a.Q = sqrt(r)*cos(theta);
    a.I = sqrt(r)*sin(theta);

    return a;
}

/*****
 *
 * Purpose: This function computes the norm of a complex number
 * Author:  Benoit Gosselin
 * Date:
 *
 *****/

float norm2c (float nbQ, float nbI)
{
    return (nbQ*nbQ + nbI*nbI);
}

/*****
 *
 * Purpose: This function implements the
 *          multiplication of two complex numbers
 * Author:  Benoit Gosselin
 * Date:
 *
 *****/

cnum multc(float a, float b, float c, float d)
{
    cnum temp;
    temp.Q = (a * c) - (b * d);
    temp.I = (a * d) + (b * c);
    return temp;
}

```

```

/*****
*
* Purpose: This function implements the square root of a complex number
*
* Author:  Benoit Gosselin
* Date:
*
*****/

cnum powc (cnum a,float n)
{
    float r, theta;
    if(n==0)
    {
        a.Q = 1;
        a.I = 0;
        return a;
    }

    r = sqrt(a.Q*a.Q + a.I*a.I);
    theta = atan(a.I/a.Q);

    theta = theta * n;

    a.Q = pow(r,n)*cos(theta);
    a.I = pow(r,n)*sin(theta);

    a=set_to_zero(a);
    return a;
}

/*****
*
* Purpose: This function implements the division of two complex numbers
* Author:  Benoit Gosselin
* Date:
*
*****/

cnum divc(float NQ,float NI,float DQ,float DI)

{
    cnum temp = multc(NQ,NI,DQ,-1.0*DI);    //multiplying by the conjugate
    float norm = norm2c(DQ,DI);
    if(norm == 0)
    {
        cout<<endl<<"divide by 0 in divc (norm = 0)";
        temp.Q=0;
        temp.I=0;
        return temp;
    }
    else
    {
        temp.Q = temp.Q / ( norm);
        temp.I = temp.I / ( norm);
        return temp;
    }
}

```

```

/*****
*
* Purpose: This function implements an Expansion recursion formula to
*          obtain the coefficients of a polynomial from its roots
* Author:  Benoit Gosselin
* Date:
* Parameters:
*   c : to store the coefficients of the polynomial
*   e : roots of the polynomial
*   m : order of the polynomial
*   n : This indice decrease at each recursion loop from n=m
*
*****/

int coeff (struct cnum *c, struct cnum e[], int n, int m)
{
    struct cnum num;
    num = multc(c[n].Q, c[n].I, e[m].Q, e[m].I);

    c[n+1].Q = c[n+1].Q - num.Q;
    c[n+1].I = c[n+1].I - num.I;

    if(n > 0)
        return coeff(&c[0], e, n-1, m);

    else
        return 0;
}

#endif

```

**DOCUMENT CONTROL DATA**

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)  
Defence Research Establishment Ottawa  
Ottawa, Ontario  
K1A 0Z4

2. SECURITY CLASSIFICATION  
(overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

Reconfigurable Digital IIR and FIR Filters (U)

4. AUTHORS (Last name, first name, middle initial)

Gosselin, B., Wilcox, C.

5. DATE OF PUBLICATION (month and year of publication of document)

November 2001

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

113

6b. NO. OF REFS (total cited in document)

12

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREO Technical Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Defence Research Establishment Ottawa  
219 Laurier Avenue  
Ottawa, Ontario K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO TR 2001-099

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

- ☒ (X) Unlimited distribution
- ☐ ( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
- ☐ ( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
- ☐ ( ) Distribution limited to government departments and agencies; further distribution only as approved
- ☐ ( ) Distribution limited to defence departments; further distribution only as approved
- ☐ ( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

The work presented in this document contributes to the ROBR (Reconfigurable Omni Band Radio) project started by the Defence Research Establishment Ottawa and the Communication Research Centre in 1997. ROBR is a testbed implementation of a reconfigurable satellite communications (satcom) terminal that makes use of a software communications architecture. Such a system can enable the use of a single ground terminal to communicate over multiple satellite communications or terrestrial links by supporting multiple standards. The ROBR hardware architecture includes a microprocessor and several digital signal processor (DSP) boards. The objective of this report is to document the work done to provide a set of reconfigurable digital filters for use in the ROBR. Five infinite impulse response (IIR) filtering modules and four finite impulse response (FIR) filtering modules have been implemented. The function of these modules is to compute the coefficients of a desired filter design. Also, IIR and FIR signal processing modules have been implemented to process digital signals using the computed coefficients. The modules have been implemented in the C programming language and are targeted for use on a DSP chip. The implementation of the modules has been verified and compared with the results obtained with the Signal Processing toolbox from MATLAB.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Digital Filter  
IIR Filter  
FIR Filter  
DSP  
Reconfigurable Filter  
Filter Coefficients



**Defence R&D Canada**

is the national authority for providing  
Science and Technology (S&T) leadership  
in the advancement and maintenance  
of Canada's defence capabilities.

**R et D pour la défense Canada**

est responsable, au niveau national, pour  
les sciences et la technologie (S et T)  
au service de l'avancement et du maintien des  
capacités de défense du Canada.



[www.drdc-rddc.dnd.ca](http://www.drdc-rddc.dnd.ca)